A COMPUTER ALGEBRA PACKAGE FOR POLYNOMIAL SEQUENCE RECOGNITION

BY

MAXIE DION SCHMIDT

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor Roy Campbell

Abstract

The software package developed in the thesis research implements functions for the intelligent guessing of polynomial sequence formulas based on user–defined expected sequence factors of the input coefficients. We present a specialized hybrid approach to finding exact representations for polynomial sequences that is motivated by the need for an automated procedures to discover the precise forms of these sums based on user guidance, or intuition, as to special sequence factors present in the formulas. In particular, the package combines the user input on the expected special sequence factors in the polynomial coefficient formulas with calls to the existing functions as subroutines that then process formulas for the remaining sequence terms already recognized by these packages. The factorization–based approach to polynomial sequence recognition is unique to this package and allows the search functions to find expressions for polynomial sums involving Stirling numbers and other special triangular sequences that are not readily handled by other software packages. The thesis contains a number of concrete, working examples of the package that are intended to both demonstrate usage and to document its current sequence recognition capabilities.

To Cameo, George, and Fred.

Table of Contents

List of Figures

List of Symbols and Notation

Notation and Conventions

$\mathbb{N}$	The set of natural numbers, $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$.
$\mathbb{Z}^+$	The set of positive integers, $\mathbb{Z}^+ = \{1, 2, 3, 4, \ldots\}$.
$\mathbb{Z}[x]$	The ring of polynomials in x with coefficients in the integers, $\mathbb{Z}$.
$\mathbb{Q}[x]$	The ring of polynomials in x with coefficients in the rational numbers, $\mathbb{Q}$.
$\mathbb{K}[x]$	The ring of polynomials in x with coefficients over the field $\mathbb{K}$.
$D_z^{(j)}$	The derivative, or differential, operator with respect to z, i.e., where $D_z^{(j)}[F(z)] \equiv F^{(j)}(z)$ denotes the j^{th} derivative of $F(z)$, provided that the j^{th} derivative of the function exists.
$\binom{n}{k}$	The binomial coefficients.
$\begin{bmatrix} n \\ k \end{bmatrix}$	The unsigned Stirling numbers of the first kind, also denoted by $(-1)^{n-k}s(n,k)$.
$\left\{ {n \atop k} \right\}$	The Stirling numbers of the second first kind, also denoted by $S(n,k)$.
$\left\langle {n \atop k} \right\rangle$	The first–order Eulerian numbers.
$\left\langle\left\langle {n \atop k} \right\rangle\right\rangle$	The second–order Eulerian numbers.
$H_n^{(r)}$	The r–order harmonic numbers, $H_n^{(r)} := \sum_{k=1}^{n} k^{-r}$, where the first–order harmonic numbers are denoted in the shorthand notation, $H_n \equiv H_n^{(1)}$.
B_n	The Bernoulli numbers.

Chapter 1

Introduction

1.1 Background and Motivation

The form of composite sequences involving the Stirling numbers of the first and second kinds are common in many applications. The Stirling number triangles arise naturally in formulas involving sums of factorial functions and in the symbolic, polynomial expansions of binomial coefficients and other factorial function variants. The Stirling and Eulerian number triangles also both frequently occur in applications involving finite sums and generating functions over non–negative powers of integers. These applications include finding closed–form expressions and formulas for generating functions over polynomial multiples of an arbitrary sequence.

Computing Formulas for the Derivatives of Stirling Number Generating Functions

If $p, k \in \mathbb{N}$, the following modified series for the generating functions for polynomial multiples of the unsigned Stirling numbers of the first kind, denoted by $\left[{n \atop k}\right]$, result in the expansions

$$\sum_{n=0}^{\infty} n^k \cdot \left[{n \atop p}\right] \frac{z^n}{n!} = \sum_{j=0}^{k} \left\{{k \atop j}\right\} z^j \cdot D_z^{(j)} \left[\frac{(-1)^p}{p!} \cdot \operatorname{Log}(1-z)^p\right] \tag{1.1}$$

$$\sum_{n=0}^{\infty} n^k \cdot \left[{n+1 \atop p+1}\right] \frac{z^n}{n!} = \sum_{j=0}^{k} \left\{{k \atop j}\right\} z^j \cdot D_z^{(j)} \left[\frac{(-1)^p}{p!} \cdot \frac{\operatorname{Log}(1-z)^p}{(1-z)}\right],$$

where the derivative operator, $D_z^{(j)}$, denotes the j^{th} derivative with respect to z of its input and the *Stirling numbers of the second kind* are denoted by $\left\{{n \atop k}\right\}$. Given enough familiarity with the Stirling numbers of the first kind and some trial and error, formulas for each of the j^{th} derivatives involved in the expansions of

(1.1) are obtained by extrapolation from the first several cases of $j \in \mathbb{N}$ to obtain the finite sums

$$D_z^{(j)}\left[\frac{(-1)^p}{p!} \cdot \text{Log}(1-z)^p\right] = \sum_{i=0}^{j} \begin{bmatrix} j \\ i \end{bmatrix} \frac{(-1)^{p-i}}{(p-i)!} \cdot \frac{\text{Log}(1-z)^{p-i}}{(1-z)^j} \tag{1.2}$$

$$D_z^{(j)}\left[\frac{(-1)^p}{p!} \cdot \frac{\text{Log}(1-z)^p}{(1-z)}\right] = \sum_{i=0}^{j} \begin{bmatrix} j+1 \\ j+1-i \end{bmatrix} \frac{(-1)^{p+j-i}}{(1-z)^{j+1}} \cdot \frac{\text{Log}(1-z)^{p-j+i}}{(p-j+i)!},$$

where the formulas in (1.2) may be regarded as polynomials in the function, $\text{Log}(1-z)$. A proof of the correctness of these formulas is then later obtained formally by induction on j.

A More Challenging Example

A more challenging, and less straightforward, example arises in attempting to find an exact, closed–form representation for the expansions of the ordinary generating function for the Stirling number sequence variant, $S_k^{(d)}(n)$, defined as in (1.3) with respect to each fixed $d, k \in \mathbb{Z}^+$.

$$S_k^{(d)}(n) := \sum_{j=1}^{n} \binom{n}{j} \begin{bmatrix} j+1 \\ k+1 \end{bmatrix} \frac{(-1)^j}{j! \cdot (j+d)} \cdot \frac{(n+d)!}{n!}, \tag{1.3}$$

The first few examples of the ordinary generating function, $\widetilde{S}_k^{(d)}(z)$, over $d \geq 2$ for the sequence defined by (1.3) are provided for reference as follows:

$$\widetilde{S}_k^{(2)}(z) = -\frac{\text{Log}\,(1-z)^{k-1}}{(1-z)^2}\left[\frac{z}{(k-1)!} + \frac{(-1+z)\,\text{Log}\,(1-z)}{k!}\right] \tag{1.4}$$

$$\widetilde{S}_k^{(3)}(z) = \frac{\text{Log}\,(1-z)^{k-2}}{(1-z)^3}\left[\frac{z^2}{(k-2)!} + \frac{z(-4+3z)\,\text{Log}\,(1-z)}{(k-1)!} + \frac{\left(2-4z+2z^2\right)\text{Log}\,(1-z)^2}{k!}\right]$$

$$\widetilde{S}_k^{(4)}(z) = -\frac{\text{Log}\,(1-z)^{k-3}}{(1-z)^4}\left[\frac{z^3}{(k-3)!} + \frac{z^2(-9+6z)\,\text{Log}\,(1-z)}{(k-2)!} + \frac{z\left(18-27z+11z^2\right)\text{Log}\,(1-z)^2}{(k-1)!} + \frac{\left(-6+18z-18z^2+6z^3\right)\text{Log}\,(1-z)^3}{k!}\right]$$

$$\widetilde{S}_k^{(5)}(z) = \frac{\text{Log}\,(1-z)^{k-4}}{(1-z)^5}\left[\frac{z^4}{(k-4)!} + \frac{z^3(-16+10z)\,\text{Log}\,(1-z)}{(k-3)!} + \frac{z^2\left(72-96z+35z^2\right)\text{Log}\,(1-z)^2}{(k-2)!} + \frac{z\left(-96+216z-176z^2+50z^3\right)\text{Log}\,(1-z)^3}{(k-1)!} + \frac{\left(24-96z+144z^2-96z^3+24z^4\right)\text{Log}\,(1-z)^4}{k!}\right].$$

Based observations of the first several cases of these generating functions in (1.4), we rewrite the expansions of these generating functions as the sum

$$\widetilde{S}_k^{(d)}(z) := \sum_{n=0}^{\infty} S_k^{(d)}(n) z^n = \left(\frac{(-1)^{d-1} \cdot \operatorname{Log}(1-z)^{k+1-d}}{(1-z)^d} \right) \times \sum_{m=0}^{d-1} \frac{\operatorname{Log}(1-z)^m \cdot z^{d-1-m}}{(k+1-d+m)!} \cdot g_m^{(d)}(z). \tag{1.5}$$

It is clear from examining the sequence data in (1.4) that the formulas for the polynomials, $g_m^{(d)}(z)$, specified in (1.5) involve a sum over factors of the Stirling numbers of the first kind and the binomial coefficients. However, finding the precise sequence inputs in the formula for these polynomials with the correct corresponding multiplier terms in the sum is not immediately obvious from the first few example cases in (1.4). We then proceed forward seeking a formula for the polynomials, $g_m^{(d)}(z)$, in the general template form of

$$g_m^{(d)}(z) = \sum_i S_1(\cdot,\cdot) \cdot \operatorname{Binom}(\cdot,\cdot) \times \operatorname{RemSeq}_1(i) \cdot \operatorname{RemSeq}_2(m + m_0 - i) \times z^i, \tag{1.6}$$

where the functions $S_1(\cdot,\cdot)$ and $\operatorname{Binom}(\cdot,\cdot)$ denote the Stirling numbers of the first kind and binomial coefficients, respectively, each over some unspecified index inputs to these sequence functions.

After a few hours of frustrating trial and error with *Mathematica*, we finally arrive at a formula for these polynomials in the form of

$$g_m^{(d)}(z) = \sum_{i=0}^{m} \begin{bmatrix} d-m+i \\ d-m \end{bmatrix} \binom{d-1}{m-i}^2 (-1)^{m-i} (m-i)! \cdot z^i. \tag{1.7}$$

The motivation for constructing the package routines in the thesis is to automate the eventual discovery of the formula in (1.7) based on user input as to the general template to the formula for these polynomials specified as in (1.6). The automated discovery of the first pair of less complicated formulas given in (1.2) is then also possible using the package.

1.2 Description of the Package

High–Level Description of the Package

The *Mathematica* package `GuessPolySequenceFormulas.m` developed as a part of the thesis research implements software routines for the intelligent guessing of polynomial sequence formulas based on user input on the expected form of the sequence formulas. These functions for sequence recognition then rely on some degree of user intuition to correctly find closed–form formulas that represent the input polynomial sequence. The logic used to construct these routines is based on factorization data for the expected sequence factors

of the input polynomial coefficient terms suggested by the user.

The template of the polynomial sequence formulas that the package aims to recognize satisfies an expansion of the general form outlined in (1.8) where $j, j_0 \in \mathbb{N}$, $r \in \mathbb{Z}^+$, and x is some (formal) polynomial variable that may assume different forms in the sequences input to the package routines.

$$\mathrm{Poly}_j(x) := \sum_{i=0}^{j+j_0} \left(\prod_{i=1}^{r} \left\| \begin{matrix} \widetilde{u}_i(j) + u_i \cdot i \\ \widetilde{\ell}_i(j) + \ell_i \cdot i \end{matrix} \right\|_i \right) \times \mathrm{RS}_1(i)\, \mathrm{RS}_2(j + j_0 - i) \cdot x^i. \tag{1.8}$$

The product of (triangular) sequences in the first term of (1.8) correspond to the factors of the expected user–defined sequences in the polynomial coefficient terms where the functions $\widetilde{u}_i(j), \widetilde{\ell}_i(j)$ are prescribed functions of the sequence index j and where the $u_i, \ell_i \in \mathbb{Z}$ are prescribed application–dependent multiples of the polynomial summation index i. The functions $\mathrm{RS}_i(\cdot)$ in the previous equation denote the coefficient remainder terms in these polynomial formulas, which should ideally correspond to comparatively simpler sequences that are already easily recognized by existing packages discussed in Section 1.3 of the thesis below. These existing packages may be called as subroutines to recognize the sequences corresponding to the remainder terms in the input polynomial coefficients after the forms of the sequence factors expected by the user are determined by the package routines.

Plan of Attack and Aims of the Thesis Research

A significant part of the work for the thesis is a "proof of concept" implementation of the logic to find polynomial sequence formulas of the form in (1.8) based on user input of the first several terms of the sequence. In this implementation, the focus of the package development is in constructing the logic to recognize the polynomial sequence formulas in the form of (1.8). For example, in the absence of obvious, or known, algorithms for the factorization of an integer by an arbitrary sequence, the implementation of this part of the algorithm employed by the package is effectively treated as an oracle within the working source code.

The construction of this type of integer factorization algorithm is motivated by the need for such algorithms in a more efficient implementation of this package. A more complete and detailed specification of these factorization routines is described in the future research topics outlined in Chapter 4. The plan is that later, once more of the machinery for generating the proposed polynomial sequence formulas is in place, optimizations to the code and the task of finding a more efficient implementation to generate the factorizations of a given integer over multiple sequence factors may be investigated further.

A listing of the *Mathematica* source code for the current revision of the package is provided in Appendix

A. Several examples of usage of the sequence recognition functions in the package, including figures of the *Mathematica* output, are given in Chapter 2. These examples provide both the working syntax of *Mathematica* programs that employ the package routines and serve to document the capabilities of the package current at the time of this thesis draft. The details and considerations encountered in the software implementation of the package are discussed in Chapter 3.

1.3 Software for Sequence Recognition

Software Packages for Sequence Recognition

There are a number of notable existing software packages and online resources geared towards guessing formulas for integer and semi–rational sequences based on the forms of the first few terms of a sequence. Notable and well–known examples include the `gfun` package for *Maple* [8], the `Rate` packages for *Mathematica*[1] [5, Appendix A], the more recently updated `Guess` package [2] for the `FriCAS` fork of *Axiom* which includes enhancements to the previous packages documented in [3], and a default, built–in function, `FindSequenceFunction`, in *Mathematica*.

There are still other software packages designed to perform related operations aimed at recognizing auxiliary properties such as identifying recurrence relations and generating functions for sequences freely available online[3]. The *Online Encyclopedia of Integer Sequences*, and its email–based `SuperSeeker` program, provide lookup access to a large database of integer sequences, including the integer–valued entries for the numerator and denominators of rational sequences such as the Bernoulli numbers. A more historical account of the development of software for sequence recognition is provided in [3, §2].

Polynomial and Summation Identities Involving the Stirling Numbers

Notice that in the absence of some underlying structure to a sequence (or satisfied by its generating function), guessing functions that attempt to find closed–form expressions for an arbitrary sequence by extrapolation from the input of its first few terms are inherently limited in obtaining a proof to verify the correctness of the formulas returned. The routines in many software packages and in the algorithms described in [7] are able to obtain computerized proofs or certificates for closed–form identities obtained from summations involving special functions. The correctness of formulas obtained by packages such as `gfun` follow if the generating

[1] See `http://www.mat.univie.ac.at/~kratt/rate/rate.html`.
[2] See `http://axiom-wiki.newsynthesis.org/GuessingFormulasForSequences`.
[3] See the complete list of *Algorithmic Combinatorics Software* on the *RISC* website at `http://www.risc.jku.at/research/combinat/software/`.

function for a sequence is *holonomic*, or equivalently, if the sequence, say S_n, itself satisfies a *P–recurrence* of the form

$$\widehat{p}_0(n) \cdot S_n + \widehat{p}_1(n) \cdot S_{n+1} + \cdots + \widehat{p}_r(n) \cdot S_{n+r} = 0, \tag{1.9}$$

whenever $n \geq n_0$ for some fixed n_0, with $r \geq 1$, and where the coefficients, $\widehat{P}_i(n) \in \mathbb{C}[n]$, are polynomials for each $0 \leq i \leq r$ [1, §B.4].

As pointed out in [2] and in [4], unlike a number of other special sequences of interest in applications, the Stirling numbers are not *holonomic*, or do not satisfy a recurrence relation of the form in (1.9), so it is reasonable to expect that existing software to guess sequence formulas should be at least somewhat limited in recognizing the exact forms of summations involving factors these sequences. The *Mathematica* package `Stirling.m` by M. Kauers is still able to find recurrences satisfied by many polynomial–like sums involving the Stirling and Eulerian number triangles in cases of many known and new summation identities. However, the example cited in Kauers' article about the package shows a seemingly simple polynomial–like summation involving the Stirling numbers of the second kind for which a recurrence relation in the form of (1.9) fails to exist [4, See §4].

This behavior offers some explanation as to the deficiency of functions like `FindSequenceFunction` in recognizing formulas for sequences involving factors of the Stirling and Bernoulli numbers. We now restrict our attention to constructing software routines that recognize formulas for the class of polynomial sums of the form in (1.8) based on intelligent guesses as to the coefficient forms input by the user. In the context, the package is intended to quickly assist the user in the discovery of formulas that arise in practice, like those motivated by the examples from Section 1.1, which we then are able to prove correct later by separate methods.

Comparisons of the Package to Existing Software Routines

The treatment of the user–defined expected sequence factors in finding formulas for input polynomial sequences is to consider these expected sequence terms as primitives in the matching formulas returned by the package. This treatment of the user–defined expected sequence factors as primitives in the search for matching formulas is analogous to the handling of the closed–form functions returned by `FindSequenceFunction` in *Mathematica*, such as for scalar or constant values, powers of (polynomials in) a variable n, factorial and gamma functions, or powers of a fixed constant, c^n.

For example, acceptable formulas returned by the package for the sequence of generating functions for polynomial powers of n may correspond to either of the sums in the following equation involving the *Stirling numbers of the second kind*, $\left\{ {n \atop k} \right\}$, or the *first–order Eulerian numbers*, $\left\langle {n \atop m} \right\rangle$, when $p \in \mathbb{N}$ and $|z| < 1$ [2, §7.4,

§6.2]:

$$\sum_{n=0}^{\infty} n^p z^n = \sum_{k=0}^{p} \left\{ {p \atop k} \right\} \frac{k! \cdot z^k}{(1-z)^{k+1}} = \frac{1}{(1-z)^{p+1}} \sum_{i=0}^{p} \left\langle {p \atop i} \right\rangle z^{i+1}.$$

The forms of sequence factors of other standard sequences, including the Stirling numbers of the first and second kinds, common variants of the binomial coefficients, $\binom{n}{k}$ and $\binom{n+m}{m}$, the Eulerian number triangles, and other triangular sequences of interest in application–specific contexts are handled similarly as primitives in the desired formulas output by the package routines.

The factorization–based approach to determine factors of expected sequences by the user in this package differs from the methods employed to recognize sequence formulas by existing sequence recognition software. Since this method relies on user direction as to what terms the sequence formulas should contain, this approach is also useful in determining formulas involving factors of difficult sequence forms that are not easily recognized by existing software packages. The package then employs a hybrid of the complementary approaches noted in [3, §1] to the search for polynomial sequence formulas. Specifically, the package routines employ existing sequence recognition functions as a subroutine to process the reminder terms in the sequence after the expected special sequence factors are identified in the coefficients of the input polynomials.

Chapter 2

The GuessPolySequenceFormulas.m Package

2.1 Features of the Package

Overview

The `GuessPolySequenceFormulas.m` package is designed to recognize formulas for polynomial sequences in one variable based on input user observations on factors of the polynomial coefficients. The public function `GuessPolynomialSequence` provided by the package attempts to perform intelligent guessing of closed–form summation representations for a polynomial sequence of elements, $p_j(x) \in \mathbb{Z}[x]$, based on the user insights as to the coefficient factors in the end formula for the sequence and the first several polynomial terms passed as input to the function. Several particular concrete examples of uses of the package to obtain formulas and other identities involving the Stirling numbers and binomial coefficients are contained in the discussions of Section 2.2 and Section 2.3 of the thesis below.

Specification of the Package Routines and Polynomial Sequence Formulas

The primary package function `GuessPolynomialSequence` provided to the user is implemented in *Mathematica* code in such a way that it is able to handle multiple coefficient factors of sequences expected by the user[1]. The focus of the examples provided as documentation for the package focus on the particular cases of "*single–factor*" and "*double–factor*" coefficient formulas for the input polynomials. In particular, the package search routines are of interest in obtaining sequence formulas corresponding to the following pair of summation formulas:

$$\mathrm{Poly}_j(x) := \sum_{i=0}^{j+j_0} \left\| \begin{matrix} \widetilde{u}_1(j) + u_1 i \\ \widetilde{\ell}_1(j) + \ell_1 i \end{matrix} \right\|_1 \times \mathrm{RS}_1(i)\,\mathrm{RS}_2(j+j_0-i) \cdot x^i \tag{2.1}$$

$$\mathrm{Poly}_j(x) := \sum_{i=0}^{j+j_0} \left\| \begin{matrix} \widetilde{u}_1(j) + u_1 i \\ \widetilde{\ell}_1(j) + \ell_1 i \end{matrix} \right\|_1 \left\| \begin{matrix} \widetilde{u}_2(j) + u_2 i \\ \widetilde{\ell}_2(j) + \ell_2 i \end{matrix} \right\|_2 \times \mathrm{RS}_1(i)\,\mathrm{RS}_2(j+j_0-i) \cdot x^i. \tag{2.2}$$

[1] See also the discussion of the running times of multiple–factor sequences in Section 3.2.

The polynomials in (2.1) and (2.2) correspond to the single–factor and double–factor sequence formula templates, respectively.

In the previous equations, $j, j_0 \in \mathbb{N}$, $u_i, \ell_i \in \mathbb{Z}$, the functions $\widetilde{u}_i(j)$ and $\widetilde{\ell}_i(j)$ denote some prescribed application–dependent functions of the sequence index, and the form of the remaining sequences in the polynomial coefficient formulas are denoted by the functions $\mathrm{RS}_1(\cdot)$ and $\mathrm{RS}_2(\cdot)$. The package formula search routines only currently handle linear functions of the summation index inputs. Also notice that it is assumed that at least one of the $RS_i(\cdot)$ sequence functions is identically one, and that a formula for the remaining function is either easily obtained by an existing sequence recognition routine such as *Mathematica*'s `FindSequenceFunction` function, or may be later identified with a relevant entry in the *Online Encyclopedia of Integer Sequences* database [9].

Special Triangular Sequence Factors Supported by the Package

The built–in subpackage `GuessSequenceData.m` included with the current package source code provides an "out of the box" implementation of several triangular sequences of interest in my research and that are important in motivating the development of this package. The rationale for the modular design in handling the triangular sequence data as a separate file within the package is explained in more detail by Section 3.1.

In the current implementation of the package, these user–specified sequences identified in the package routines include factors of the (signed and unsigned) Stirling number triangles, variations of triangular sequences derived from the binomial coefficients, and the first and second–order Eulerian number triangles defined recursively as in [2, §6.1, 6.2] [6, *c.f.* §26.8, 26.14]. Each of these respective sequences correspond to special cases of the following triangular recurrence relation where $\alpha, \beta, \gamma, \alpha', \beta', \gamma' \in \mathbb{Z}$ [2, §5, §6.1–6.2]:

$$\left\|{n \atop k}\right\| = (\alpha n + \beta k + \gamma) \left\|{n-1 \atop k}\right\| + (\alpha' n + \beta' k + \gamma') \left\|{n-1 \atop k-1}\right\| + [n = k = 0]_\delta \,.$$

Mathematica provides several standard, built–in functions for the (signed) Stirling numbers of the first and second kinds, and for the binomial coefficients. The related *Mathematica* package `Stirling.m` authored by Manuel Kauers [2] further extends the default functions for the Stirling numbers and defines additional functions that implement the Eulerian number triangles of both orders [4].

[2] See also the package documentation at `http://www.risc.jku.at/research/combinat/software/ergosum/RISC/Stirling.html`.

Some Restrictions on the Form of the Input Polynomials

The package function `GuessPolynomialSequence` is designed to find formulas for polynomials, $p_j(x)$, whose coefficients are integer–valued. The guessing function is, however, able to find formulas for semi–rational polynomial sequences in $\mathbb{Q}[x]$ provided that the first several terms of the sequence input to the function `GuessPolynomialSequence` are normalized by a user guess function, $U_{\text{guess}}(j, i)$, as described in Section 2.2.3 of the thesis below. The difficulties in handling formulas for polynomials with rational coefficients arise in determining strictly integer–valued factors of rational–valued coefficient forms. These implementation issues are outlined in Section 3.3. Several suggestions for transformations that pre–process polynomials with rational coefficients are also suggested in the section as features to be implemented in a future revision of the package.

2.2 Package Usage and Examples

2.2.1 Installation

The package requires a working installation of *Mathematica* and a copy of the two source files `GuessPolySequenceFormulas.m` and `GuessSequenceData.m` provided by the respective listings of Appendix A. To load the package under Linux, suppose that the package files are located in `~/guess-polys-pkg`. The package is then loaded by running

```
<<"~/guess-polys-pkg/GuessPolySequenceFormulas.m"
```

A graphical summary of the short description and revision information for the package is printed when the package is successfully loaded from within a *Mathematica* notebook.

2.2.2 Typical Usage

The examples given in this section illustrate both the syntax and utility of the sequence recognition routines provided by the package functions. Notice that the formulas returned by the function are pure functions in *Mathematica* with three ordered parameters: 1) The polynomial sequence index; 2) An input variable that denotes the summation index of the formula; and 3) A parameter that specifies the polynomial variable.

The graphical printing of the formula data provided in the figures given in this section is disabled by setting the runtime option `PrintFormulas->False`. The runtime option `FSFFunction` is also available to replace the default *Mathematica* function `FindSequenceFunction` by an alternate sequence handling function to process the formulas for the remaining sequences in the polynomial coefficient terms, as well as

the formulas for the coefficient indices in the polynomial index j and the upper index of summation in (2.1) and (2.2).

The expected coefficient factors in the formulas returned by the function defaults to the form of the (unsigned) triangle of Stirling numbers of the first kind unless otherwise specified by resetting the runtime option `SequenceFactors->{"S1"}`. Notice that the value of the start index of the sequence passed to the function `GuessPolynomialSequence` defaults to the runtime option value of `StartIndex->1`. A complete listing of the runtime configuration options defined for use with the package are found starting on line 125 of the source code given in Appendix A.2. The most common and useful of these option settings are documented in the examples below and in the sections of this chapter.

Coefficient Factors Involving the Stirling Numbers of the First Kind

Consider the following pair of sums resulting from the expansions of the binomial coefficients as polynomials in n

$$\begin{aligned}\binom{n}{k} &= \frac{n^{\underline{k}}}{k!} = \frac{1}{k!} \times n \cdot (n-1) \cdot (n-2) \cdots (n-k+1) \\ &= \frac{1}{k!} \times \sum_{i=0}^{k} \begin{bmatrix} k \\ i \end{bmatrix} (-1)^{k-i} n^i \qquad (2.3) \\ \binom{n+m}{m} &= \frac{n^{\overline{m+1}}}{n \cdot m!} = \frac{1}{m!} \times (n+1) \cdot (n+2) \cdots (n+m-1) \cdot (n+m) \\ &= \frac{1}{m!} \times \sum_{i=0}^{m} \begin{bmatrix} m+1 \\ i+1 \end{bmatrix} n^i, \qquad (2.4)\end{aligned}$$

where $n^{\underline{k}}$ denotes the *falling factorial function* and $n^{\overline{m}}$ is the *rising factorial function* in the respective expansions of the previous equations [2, §2.6; §5.1] [6, *c.f.* §26.1].

The sums for the binomial coefficient expansions involving the Stirling numbers in each of (2.3) and (2.4) are known closed–form identities for the rising and falling factorial functions, respectively, stated in [2, §6.1] [6, *c.f.* §26.8]. To see how the package can assist a user in rediscovering these identities, consider the respective *Mathematica* outputs given in Figure 2.1 and Figure 2.2. A related example involving the polynomial sequence in (2.3) is shown in Figure 2.7.

Multiple Formulas Derived from Symmetric Sequences

The package sequence recognition routines are able to find formulas for polynomials involving the *binomial coefficients*, $\binom{n}{k}$, and the *first–order Eulerian numbers*, $\left\langle {n \atop m} \right\rangle$. These sequences both have symmetry in each row of the corresponding triangles that satisfy the following pair of reflection identities where $n, k, m \in \mathbb{N}$

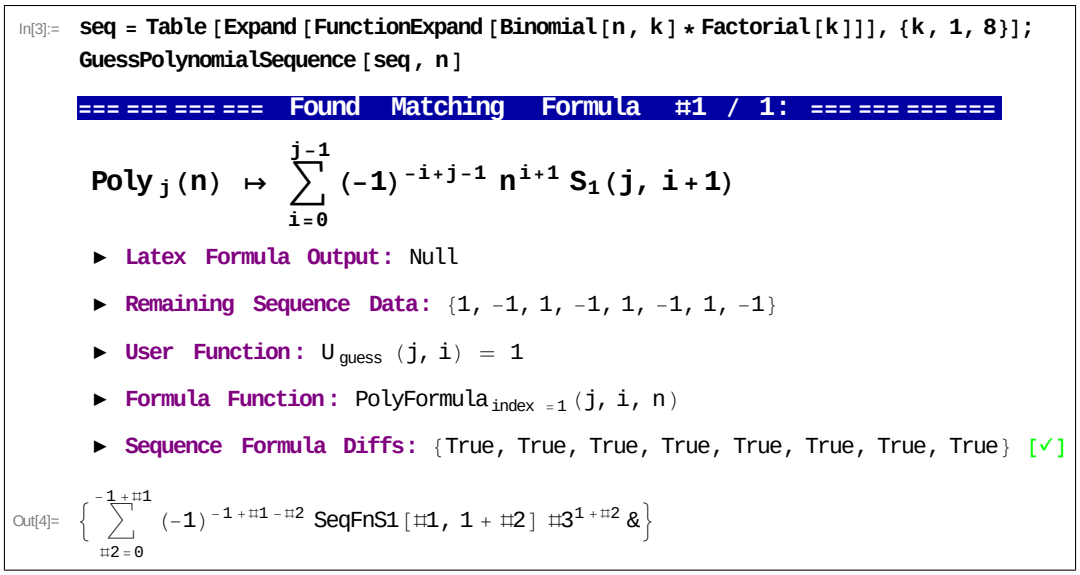

Figure 2.1: Computing a Polynomial Formula for the Falling Factorial Function

[2, §5, §6.2]:

$$\binom{n}{k} = \binom{n}{n-k} \qquad \text{and} \qquad \left\langle {n \atop m} \right\rangle = \left\langle {n \atop n-1-m} \right\rangle.$$

The examples given in this section demonstrate the multiple formulas obtained by the package for polynomial sequences involving these triangles that result from the coefficient symmetry noted in the forms of the previous equation.

The first example corresponds to an identity involving squares of the binomial coefficients and the *derivative operator*, $D^{(j)}[F(z)] \equiv F^{(j)}(z)$, of a function, $F(z)$, whose j^{th} derivative with respect to z exists for some $j \in \mathbb{N}$. In particular, suppose that the function $F(z)$ denotes the ordinary generating function of the sequence, $\langle f_n \rangle$, and the function has j^{th} derivatives of orders $j \in [0, d] \subseteq \mathbb{N}$. Then for $d \in \mathbb{Z}^+$, the generating function for the modified sequence, $\langle \frac{(n+d)!}{n!} f_n \rangle$, satisfies the formula

$$\sum_{n=0}^{\infty} \frac{(n+d)!}{n!} f_n z^n = \sum_{n=0}^{\infty} (n+1) \cdots (n+d) \times f_n z^n = \sum_{i=0}^{d} \binom{d}{i}^2 (d-i)! \times z^i D^{(i)}[F(z)]. \tag{2.5}$$

Notice that a proof of the formula given in (2.5) follows easily by induction on $d \geq 1$. A user may obtain the first several values of this sequence empirically by evaluating *Mathematica*'s `GeneratingFunction` for the modified sequence terms over the first few values of $d \geq 1$. Figure 2.3 shows a use of the package in guessing a formula for (2.5) where the polynomial variable (w^i in the figure listing) corresponds to the operator form of $z^i D^{(i)}$, and where the *Pochhammer symbol*, $(1)_{j-i} \equiv (j-i)!$.

The next example in this section corresponds to the sequence of ordinary generating functions for poly-

```
In[13]:= seq = Table[Expand[FunctionExpand[Binomial[n + m, m] * Factorial[m]]], {m, 0, 8}];
         GuessPolynomialSequence[seq, n, StartIndex → 0, PrintLaTeXFormulas → False]

=== === === ===  Found  Matching  Formula  #1 / 1:  === === === ===

                    j
Poly_j(n) ↦  Σ   n^i S_1(j + 1, i + 1)
                   i=0

► Remaining Sequence Data: {1, 1, 1, 1, 1, 1, 1, 1, 1}

► User Function: U_guess (j, i) = 1

► Formula Function: PolyFormula_index = 1 (j, i, n)

► Sequence Formula Diffs: {True, True, True, True, True, True, True, True, True} [✓]

Out[14]= { Σ   SeqFnS1[1 + #1, 1 + #2] #3^#2 & }
         #2=0
```

Figure 2.2: Computing a Polynomial Formula for the Rising Factorial Function

nomial powers of n, $\sum_{n\geq 0} n^m z^n$ for $m \in \mathbb{N}$. These generating functions satisfy well–known polynomial identities involving the *Stirling numbers of the second kind*, $\left\{ {n \atop k} \right\}$, and the first–order Eulerian numbers, $\left\langle {n \atop m} \right\rangle$, stated as follows [2, *c.f.* §7.4]:

$$\sum_{n=0}^{\infty} n^m z^n = \sum_{k=0}^{m} \left\{ {m \atop k} \right\} \frac{k! \cdot z^k}{(1-z)^{k+1}} = \frac{1}{(1-z)^{m+1}} \sum_{i=0}^{m} \left\langle {m \atop i} \right\rangle z^{i+1}. \tag{2.6}$$

The second example cited in this section focuses on the second expansion in (2.6) given in terms of the Eulerian number triangle. Figure 2.4 shows the output of the package on the second polynomial sequence scaled by a multiple of $(1-z)^{m+1}$. As with the first example, the row–wise symmetry in the Eulerian number triangle results in the two separate formulas in the figure.

Examples of Two–Factor Polynomial Sequence Formulas

The examples cited in this section correspond to the two–factor polynomial sequence formulas in the form of (2.2). The first example given in Figure 2.5 shows the output of the package function `GuessPolynomialSequence` for a formula involving the Stirling numbers of the first and second kinds. The second example given in Figure 2.6 shows the pair of formulas output for a sequence formula involving the Stirling numbers of the first kind and the binomial coefficients. The use of the runtime option `IndexOffsetPairs` in both of these examples is explained in more detail by Section 2.2.4.

```
sumFn[d_, w_] := Sum[(Binomial[d, i]^2) * Factorial[d - i] * Power[w, i], {i, 0, d}]
(***)
seq = Table[sumFn[d, w], {d, 1, 6}];
GuessPolynomialSequence[seq, w, SequenceFactors → {"Binom2"}]
```

=== === === === Found Matching Formula #1 / 2: === === === ===

$$\text{Poly}_j(w) \mapsto \sum_{i=0}^{j} w^i \binom{j}{j-i}^2 (1)_{j-i}$$

- Latex Formula Output: Null
- Remaining Sequence Data: {1, 1, 2, 6, 24, 120, 720}
- User Function: $U_{\text{guess}}(j, i) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=1}(j, i, w)$
- Sequence Formula Diffs: {True, True, True, True, True, True} [✓]

=== === === === Found Matching Formula #2 / 2: === === === ===

$$\text{Poly}_j(w) \mapsto \sum_{i=0}^{j} w^i \binom{j}{i}^2 (1)_{j-i}$$

- Latex Formula Output: Null
- Remaining Sequence Data: {1, 1, 2, 6, 24, 120, 720}
- User Function: $U_{\text{guess}}(j, i) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=2}(j, i, w)$
- Sequence Formula Diffs: {True, True, True, True, True, True} [✓]

$$\left\{ \sum_{\#2=0}^{\#1} \text{Binomial}[\#1, \#1 - \#2]^2 \, \text{Pochhammer}[1, \#1 - \#2] \, \#3^{\#2} \,\&, \right.$$

$$\left. \sum_{\#2=0}^{\#1} \text{Binomial}[\#1, \#2]^2 \, \text{Pochhammer}[1, \#1 - \#2] \, \#3^{\#2} \,\& \right\}$$

Figure 2.3: A Formula Involving Derivative Operators and Squares of the Binomial Coefficients

2.2.3 Specifying a User Guess Function

The guessing routines implemented in the package rely on some intuition on the part of the user to determine a general template for the end formulas for an input polynomial sequence with coefficients over the integers. The user may specify an additional "*user guess function*" that is employed by the package to pre–process the coefficients of the polynomial sequence terms passed to the function `GuessPolynomialSequence` . This construction allows semi–rational, and even non–polynomial functions in the input variable to be processed by the package functions.

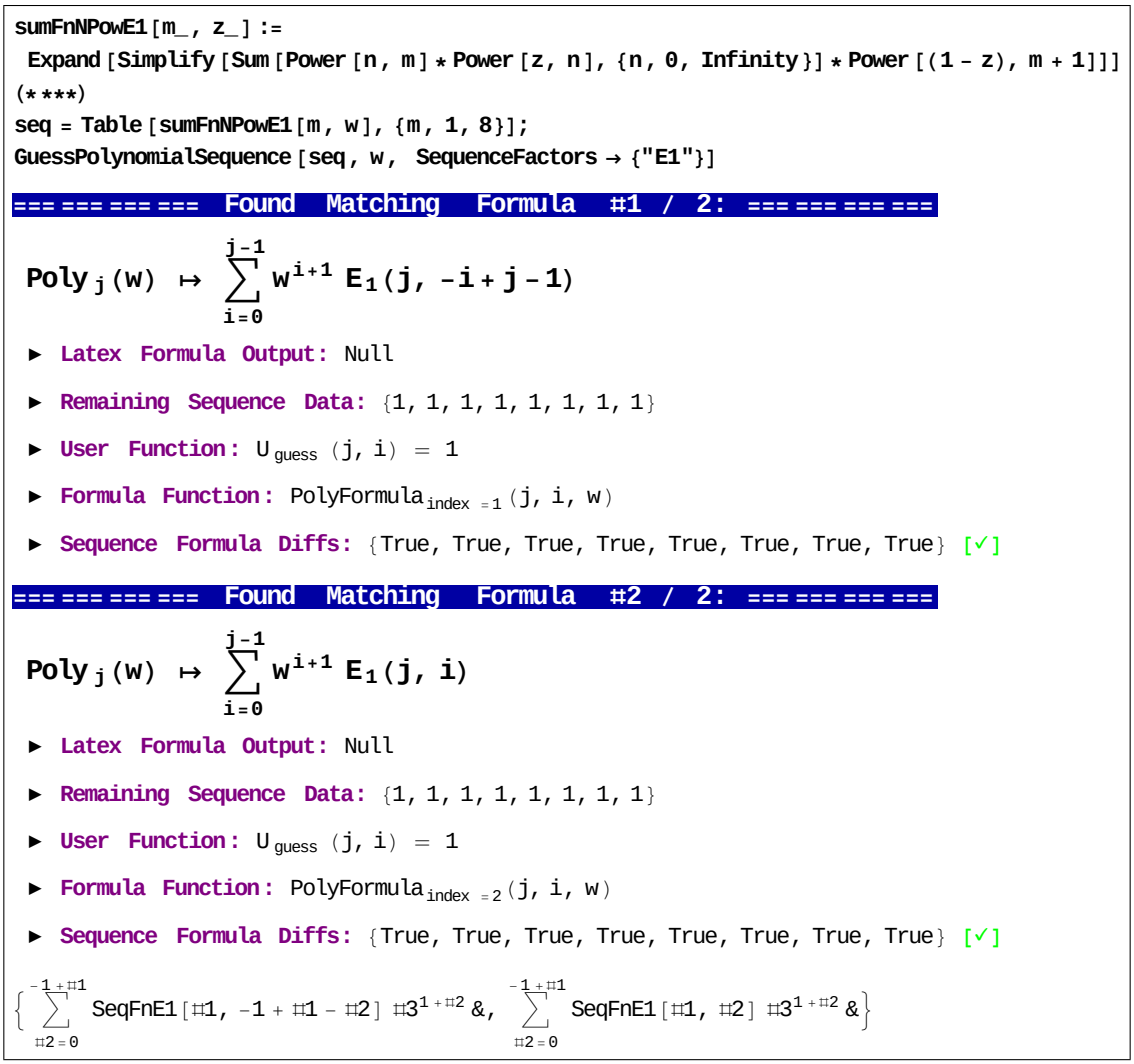

Figure 2.4: Ordinary Generating Functions of Polynomial Powers

A Second Formula for the Falling Factorial Function

A first example of the syntax for guessing the polynomial expansions of the binomial coefficient identity from (2.3) is provided in Figure 2.7. Notice that this example is similar to the first form of the sequence formula computed by the package in Figure 2.1, except that in this case the input sequence is not normalized by a factor of k to make the polynomial coefficients strictly integer–valued. A similar computation is employed to discover an analogous sum for the non–normalized sequence formula corresponding to the rising factorial function from Figure 2.2.

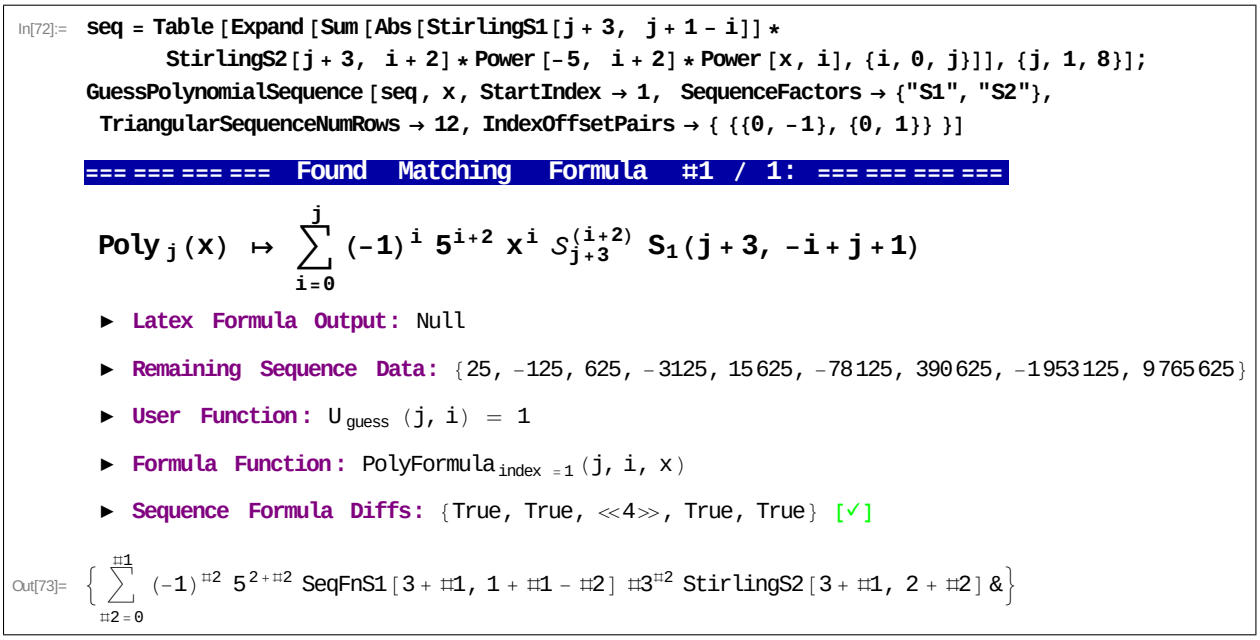

Figure 2.5: A Double–Factor Sequence Example Involving the Stirling Number Triangles

An Exponential Generating Function for the Binomial Coefficients

A sequence of exponential generating functions for the symmetric form of the binomial coefficients, $\binom{n+m}{m}$, taken over $m \in \mathbb{N}$ satisfies the formula given in the following equation [2, *c.f.* §7.2]:

$$\mathrm{EGF}_z\left(\frac{1}{(1-z)^{m+1}}\right) \equiv \sum_{n=0}^{\infty} \binom{n+m}{n} \frac{z^n}{n!} \equiv \sum_{s=0}^{m} \binom{m}{s} \frac{e^z \cdot z^s}{s!}. \tag{2.7}$$

A proof of this identity is given using *Vandermonde's convolution* identity for the binomial coefficients [2, §Table 174; §5.2; *c.f.* eq. (5.22)]. Figure 2.8 shows a use of the package to guess the formula in (2.7) by providing a user guess function that effectively removes the factor of e^z in the expected formula, and that cancels out the coefficient factors of $1/s!$ to produce an input sequence with integer coefficients.

2.2.4 Possible Issues

Inputting an Insufficient Number of Sequence Elements

There are a couple of issues that can arise in running the package routines when too few values of the sequence are passed to the `GuessPolynomialSequence` function. The first of these is that `FindSequenceFunction` may require a lower bound on the number of sequence values necessary to compute formulas for the remaining sequence terms. This can occur, for example, when the remaining sequence is a polynomial in the summation

```
In[74]:= seq = Table[Expand[Sum[Abs[StirlingS1[j + 3, j + 1 - i]] *
        Binomial[j + 3, i + 2] * Power[-5, i + 2] * Power[x, i], {i, 0, j}]], {j, 1, 8}];
GuessPolynomialSequence[seq, x, StartIndex → 1, SequenceFactors → {"S1", "Binom"},
 TriangularSequenceNumRows → 12,
 IndexOffsetPairs → { {{0, -1}, {0, 1}}, {{0, -1}, {0, -1}} }, DisplayVars → {m, k}]
```

=== === === === Found Matching Formula #1 / 2: === === === ===

$$\text{Poly}_m(x) \mapsto \sum_{k=0}^{m} (-1)^k \, 5^{k+2} \, x^k \binom{m+3}{k+2} S_1(m+3, -k+m+1)$$

- Latex Formula Output: Null
- Remaining Sequence Data: {25, -125, 625, -3125, 15625, -78125, 390625, -1953125, 9765625}
- User Function: $U_{guess}(m, k) = 1$
- Formula Function: $\text{PolyFormula}_{index=1}(m, k, x)$
- Sequence Formula Diffs: {True, True, <<4>>, True, True} [✓]

=== === === === Found Matching Formula #2 / 2: === === === ===

$$\text{Poly}_m(x) \mapsto \sum_{k=0}^{m} (-1)^k \, 5^{k+2} \, x^k \binom{m+3}{-k+m+1} S_1(m+3, -k+m+1)$$

- Latex Formula Output: Null
- Remaining Sequence Data: {25, -125, 625, -3125, 15625, -78125, 390625, -1953125, 9765625}
- User Function: $U_{guess}(m, k) = 1$
- Formula Function: $\text{PolyFormula}_{index=2}(m, k, x)$
- Sequence Formula Diffs: {True, True, <<4>>, True, True} [✓]

```
Out[75]= { Sum[(-1)^#2 5^(2+#2) Binomial[3 + #1, 2 + #2] SeqFnS1[3 + #1, 1 + #1 - #2] #3^#2 &, {#2, 0, #1}],
           Sum[(-1)^#2 5^(2+#2) Binomial[3 + #1, 1 + #1 - #2] SeqFnS1[3 + #1, 1 + #1 - #2] #3^#2 &, {#2, 0, #1}] }
```

Figure 2.6: A Double–Factor Sequence Example Involving the Stirling Numbers of the First Kind and the Binomial Coefficients

index.

Another quirk of *Mathematica*'s built–in `FindSequenceFunction` is that it may return a sequence formula matching a recurrence relation that is actually accurate for the few sequence elements input to the function. An example of this behavior is illustrated by the output given in Figure 2.9. In most cases, the problem is resolved by simply passing more polynomials from the sequence, usually at least 6, but possibly 8 or more elements from the sequence. The package is configured to warn users when less than 6 initial terms are input to the function with no matching formulas.

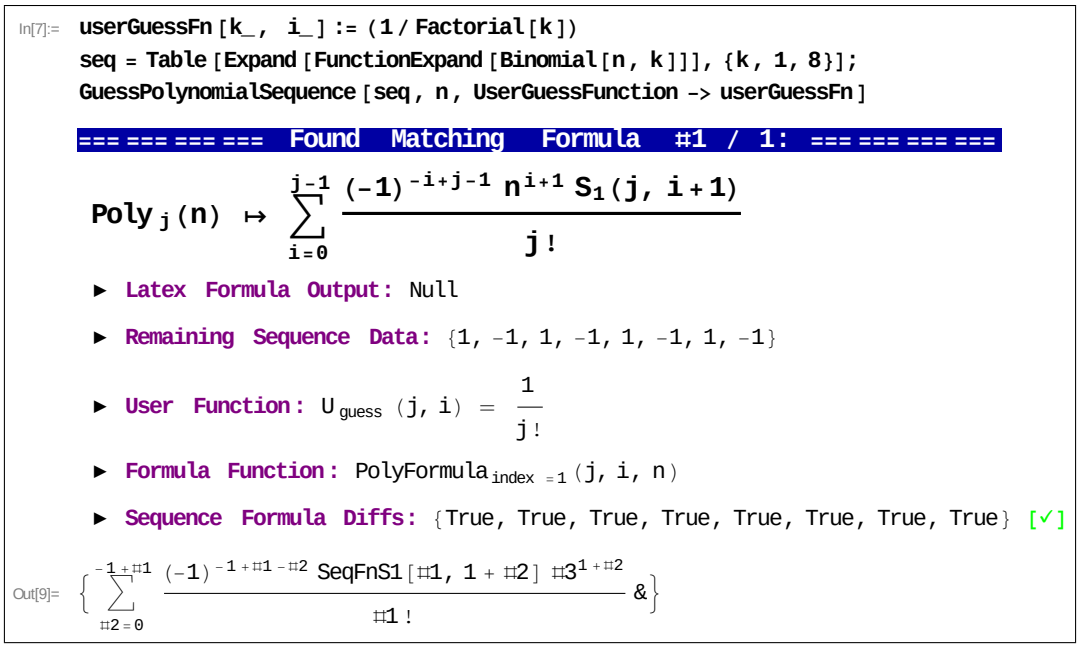

Figure 2.7: Computing a Formula for the Falling Factorial Function Using a User Guess Function

Number of Rows for the Expected Triangular Sequence Factors

In some cases, the package functions may not be able to obtain a formula for an input sequence due to an insufficient setting for the number of rows to consider for the expected triangular sequence factors. The runtime option to change the number of rows used to detect the factors of the expected triangular sequence is `TriangularSequenceNumRows` (the current default setting is `TriangularSequenceNumRows->12`).

Figure 2.10 provides an example involving the Stirling numbers of the second kind where the upper index of the sequence depends quadratically on the polynomial index j. In this example, the package routines are unable to obtain a formula when the runtime option is reset to `TriangularSequenceNumRows->24`, but correctly finds the sequence formula by setting the option to the higher value of `TriangularSequenceNumRows->72`.

Notice that choosing a significantly higher default setting for this option may result in much slower running times, especially if the expected triangular sequence factors contain a large number of 1–valued entries, for example, as in the Stirling numbers of the first kind, binomial coefficient, and first–order Eulerian number triangles.

```
In[33]:= userGuessFn[k_, s_] := (Exp[z] / Factorial[s])
         seq =
           Table[Simplify[Sum[Binomial[n + k, k] * Power[z, n] / Factorial[n], {n, 0, Infinity}]], {k, 0, 8}];
         GuessPolynomialSequence[seq, z, StartIndex → 0, SequenceFactors → {"Binom"},
          UserGuessFunction -> userGuessFn]
```

=== === === === Found Matching Formula #1 / 2: === === === ===

$$\text{Poly}_j(z) \mapsto \sum_{i=0}^{j} \frac{e^z z^i \binom{j}{j-i}}{i!}$$

- Latex Formula Output: Null
- Remaining Sequence Data: {1, 1, 1, 1, 1, 1, 1, 1, 1}
- User Function: $U_{guess}(j, i) = \frac{e^z}{i!}$
- Formula Function: $\text{PolyFormula}_{index=1}(j, i, z)$
- Sequence Formula Diffs: {True, True, <<5>>, True, True} [✓]

=== === === === Found Matching Formula #2 / 2: === === === ===

$$\text{Poly}_j(z) \mapsto \sum_{i=0}^{j} \frac{e^z z^i \binom{j}{i}}{i!}$$

- Latex Formula Output: Null
- Remaining Sequence Data: {1, 1, 1, 1, 1, 1, 1, 1, 1}
- User Function: $U_{guess}(j, i) = \frac{e^z}{i!}$
- Formula Function: $\text{PolyFormula}_{index=2}(j, i, z)$
- Sequence Formula Diffs: {True, True, <<5>>, True, True} [✓]

```
Out[35]= { Sum_{#2=0}^{#1} e^z Binomial[#1, #1 - #2] #3^#2 / #2! &,
           Sum_{#2=0}^{#1} e^z Binomial[#1, #2] #3^#2 / #2! & }
```

Figure 2.8: An Exponential Generating Function for the Binomial Coefficients

Handling Long Running Times with Multiple Sequence Factors

The package function `GuessPolynomialSequence` is able to return sequence formulas in the single–factor form given in (2.1) in a reasonable amount of running time. As suggested in the double–factor sequence examples of the form in (2.2) from Figure 2.5 and Figure 2.6, the runtime option `IndexOffsetPairs` is needed to speed–up the running time for the computations involved in these sequence cases.

The `IndexOffsetPairs` option is defined as a list of lists of the form

$$\{\{u_1, \ell_1\}, \{u_2, \ell_2\}, \ldots, \{u_r, \ell_r\}\}, \tag{2.8}$$

```
In[65]:= sumFnE1[j_, w_] := Sum[(SeqFnE1[j, i]) * Power[-1, j - i] * Power[w, i], {i, 0, j}]
(***)
seq = Table[sumFnE1[j, w], {j, 1, 6}];
GuessPolynomialSequence[seq, w, StartIndex → 1, SequenceFactors → {"E1"}, LimitFormulaCount → 2]
```

=== === === === Found Matching Formula #1 / 34 : === === === ===

$$\text{Poly}_j(w) \mapsto \sum_{i=0}^{j-1} (-1)^{j-i} w^i$$

$$E_1\left(\text{DifferenceRoot}\left[\{\dot{y}, \dot{n}\} \mapsto \{-\dot{y}(\dot{n}) + \dot{y}(\dot{n}+1) - 1 = 0, \dot{y}(1) = 0, \dot{y}(2) = 2\}\right][j], -i + j - 1\right)$$

- Latex Formula Output: Null
- Remaining Sequence Data: {-1, 1, -1, 1, -1, 1}
- User Function: $U_{\text{guess}}(j, i) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=1}(j, i, w)$
- Sequence Formula Diffs: {True, True, <<2>>, True, True} [✓]

=== === === === Found Matching Formula #2 / 34 : === === === ===

$$\text{Poly}_j(w) \mapsto \sum_{i=0}^{j-1} (-1)^{j-i} w^i \, E_1(j, -i + j - 1)$$

- Latex Formula Output: Null
- Remaining Sequence Data: {-1, 1, -1, 1, -1, 1}
- User Function: $U_{\text{guess}}(j, i) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=2}(j, i, w)$
- Sequence Formula Diffs: {True, True, <<2>>, True, True} [✓]

```
Out[67]= { Sum_{#2=0}^{-1+#1} (-1)^(#1-#2)
    SeqFnE1[DifferenceRoot[Function[{ẏ, ṅ}, {-1 - ẏ[ṅ] + ẏ[1 + ṅ] == 0, ẏ[1] == 0, ẏ[2] == 2}]][#1],
     -1 + #1 - #2] #3^#2 &, Sum_{#2=0}^{-1+#1} (-1)^(#1-#2) SeqFnE1[#1, -1 + #1 - #2] #3^#2 & }
```

Figure 2.9: Troubleshooting an Insufficient Number of Sequence Elements

where $r \geq 1$ denotes the expected number of sequence factors involved in the search for the sequence formulas by `GuessPolynomialSequence`. In the examples cited in Figure 2.5, Figure 2.6, and in the template form of (2.2), the value of r corresponds to $r := 2$. For a fixed choice of $r \geq 1$, each element of the list defined by `IndexOffsetPairs` passed in the form of (2.8) corresponds to a search for a sequence formula of the form

$$\text{Poly}_j(x) := \sum_{i=0}^{j+j_0} \left(\prod_{i=1}^{r} \left\| \begin{matrix} \widetilde{u}_i(j) + u_i i \\ \widetilde{\ell}_i(j) + \ell_i i \end{matrix} \right\|_i \right) \times \text{RS}_1(i)\, \text{RS}_2(j + j_0 - i) \cdot x^i.$$

Thus resetting the value of this option at runtime can speed–up the search for matching formulas in the

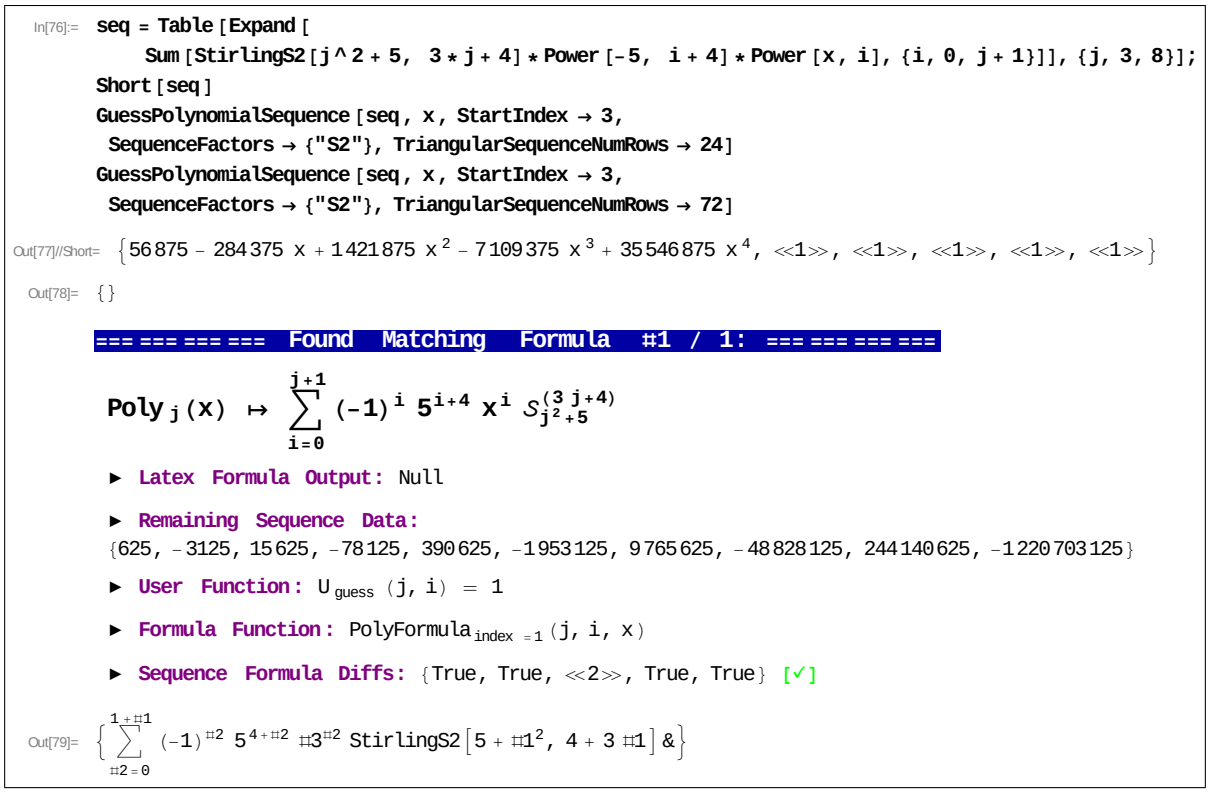

Figure 2.10: Troubleshooting Runtime Settings of the `TriangularSequenceNumRows` Option

cases of multiple expected sequence factors, especially compared to the number of index offset pairs resulting from the default enumeration of these pair values.

2.3 More Examples of Sequence Types Recognized by the Package

The examples cited in this section are intended to document further forms of the polynomial sequence types that the package is able to recognize. These examples include handling polynomial sequence formulas that depend on arithmetic progressions of indices, coefficients that contain symbolic data, and examples of sequence formulas obtained by the package routines when the expected sequence factors do not depend on the summation index, i.e., when the factors only depend on the polynomial sequence index.

Arithmetic Progressions of Coefficient Indices

The package function `GuessPolynomialSequence` can be configured to search for sequence formulas involving arithmetic progressions of the summation index, $f(j)+ai$, for values besides $a := \pm 1$ by resetting the runtime

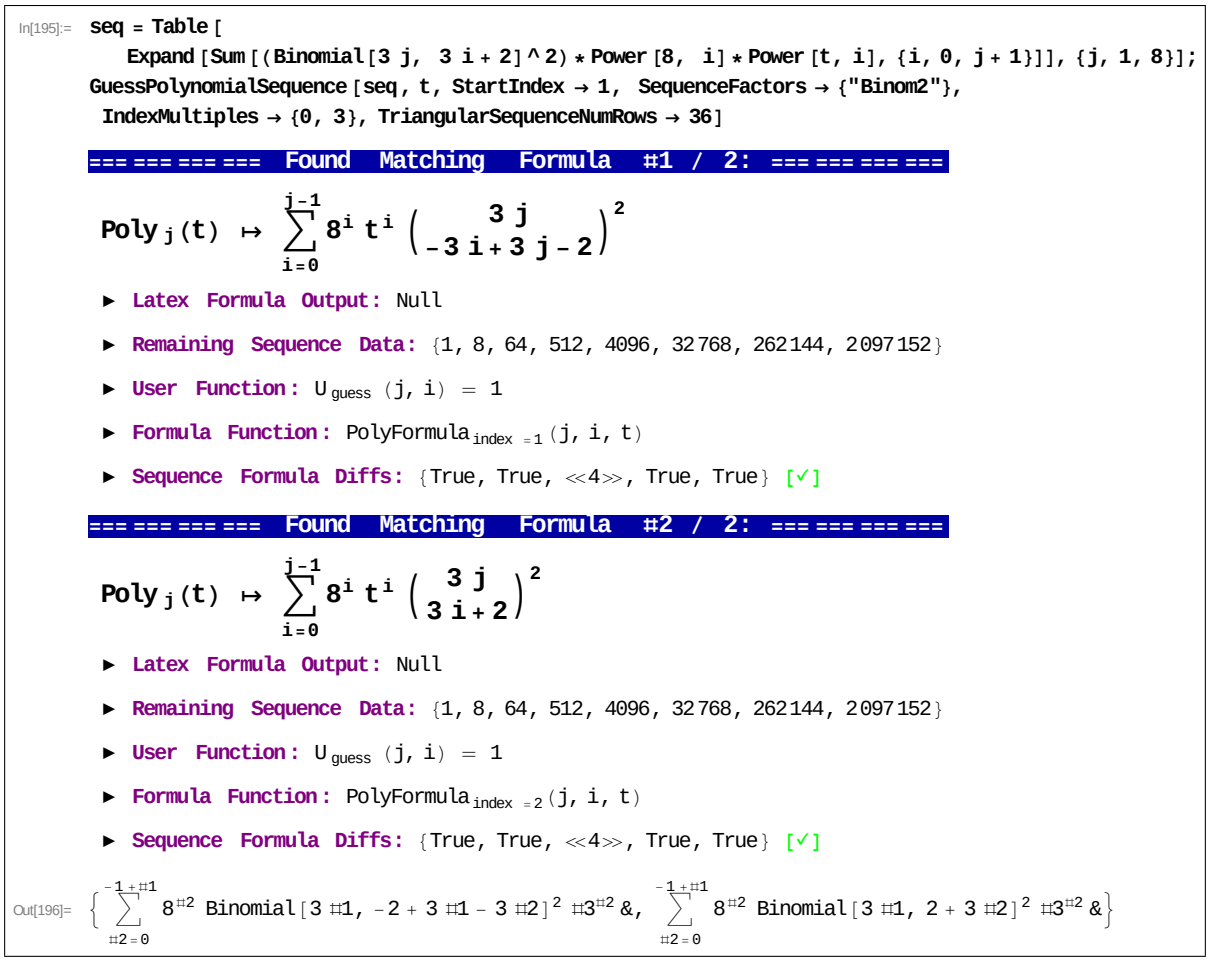

Figure 2.11: Handling Arithmetic Progressions of Indices

option `IndexMultiples`. The default setting of this option is `IndexMultiples->{0,1}`. Figure 2.11 provides an example of recognizing sequence formulas involving squares of the binomial coefficients where the upper index of the triangle does not depend on the summation index (a setting of $a := 0$) and where the lower triangle index involves an arithmetic progression of the summation index with $a := \pm 3$.

Related sequence formulas are recognized by setting the runtime value of this option to a list of test values that is some subset of the natural numbers. Notice that if the list of values for the option `IndexMultiples` does not contain 0, the package routines will not find formulas like those given in Figure 2.11 where the upper index of the expected triangle factors only depends on the polynomial sequence index (j in the figure examples).

```
In[102]:= seq = Table[Expand[Sum[Binomial[j + 3, j + 1 - i] * Power[-5 * a, i + 2] *
        Power[6 * b, i + 1] * Power[c, i + 3] * Power[z, i], {i, 0, j}]], {j, 3, 8}];
GuessPolynomialSequence[seq, z, StartIndex → 3, SequenceFactors → {"Binom"},
 DisplayVars → {n, k}, AllowSymbolicData → True, LimitFormulaCount → 1]
```

=== === === === Found Matching Formula #1 / 2: === === === ===

$$\text{Poly}_n(z) \mapsto \sum_{k=0}^{n} a^2\, b\, c^3\, 5^{k+2}\, 6^{k+1}\, z^k \binom{n+3}{-k+n+1} (-a\, b\, c)^k$$

- Latex Formula Output: Null
- Remaining Sequence Data: {150 a^2 b c^3, -4500 a^3 b^2 c^4, 135000 a^4 b^3 c^5, -4050000 a^5 b^4 c^6, 121500000 a^6 b^5 c^7, -3645000000 a^7 b^6 c^8, 109350000000 a^8 b^7 c^9, -3280500000000 a^9 b^8 c^10, 98415000000000 a^10 b^9 c^11}
- User Function: $U_{\text{guess}}(n, k) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=1}(n, k, z)$
- Sequence Formula Diffs: {True, True, True, True, True, True} [✓]

```
Out[103]= { Sum_{#2=0}^{#1} 5^(2+#2) 6^(1+#2) a^2 b c^3 (-a b c)^#2 Binomial[3 + #1, 1 + #1 - #2] #3^#2 & }
```

Figure 2.12: Recognizing Sequence Formulas Involving Symbolic Coefficients

```
In[104]:= seq = Table[Expand[Sum[StirlingS2[j + 3, j + 1 - i] *
        Power[-5 * r, i + 2] * Power[q, i^2] * Power[y, i], {i, 0, j}]], {j, 1, 8}];
GuessPolynomialSequence[seq, y, SequenceFactors → {"S2"}, FullSimplifyFormulas → True,
 DisplayVars → {n, k}, AllowSymbolicData → True]
```

=== === === === Found Matching Formula #1 / 1: === === === ===

$$\text{Poly}_n(y) \mapsto \sum_{k=0}^{n} (-1)^k\, 5^{k+2}\, q^{k^2}\, r^{k+2}\, y^k\, S_{n+3}^{(-k+n+1)}$$

- Latex Formula Output: Null
- Remaining Sequence Data: {25 r^2, -125 q r^3, 625 q^4 r^4, -3125 q^9 r^5, 15625 q^16 r^6, -78125 q^25 r^7, 390625 q^36 r^8, -1953125 q^49 r^9, 9765625 q^64 r^10}
- User Function: $U_{\text{guess}}(n, k) = 1$
- Formula Function: $\text{PolyFormula}_{\text{index}=1}(n, k, y)$
- Sequence Formula Diffs: {True, True, True, True, True, True, True, True} [✓]

```
Out[105]= { Sum_{#2=0}^{#1} (-1)^#2 5^(2+#2) q^#2^2 r^(2+#2) #3^#2 StirlingS2[3 + #1, 1 + #1 - #2] & }
```

Figure 2.13: A Second Formula Involving Square Index Powers of Symbolic Coefficients

Formulas Involving Symbolic Coefficient Data

The function `GuessPolynomialSequence` can be configured to search for formulas where the input coefficients of the polynomial sequence contain non–numeric factors of symbolic data through the runtime option

AllowSymbolicData. Figure 2.12 and Figure 2.13 provide examples of sequence formulas involving non–numeric, symbolic terms, named a, b, c, q, r, that are recognized by the package by passing AllowSymbolicData->True to the GuessPolynomialSequence function at runtime.

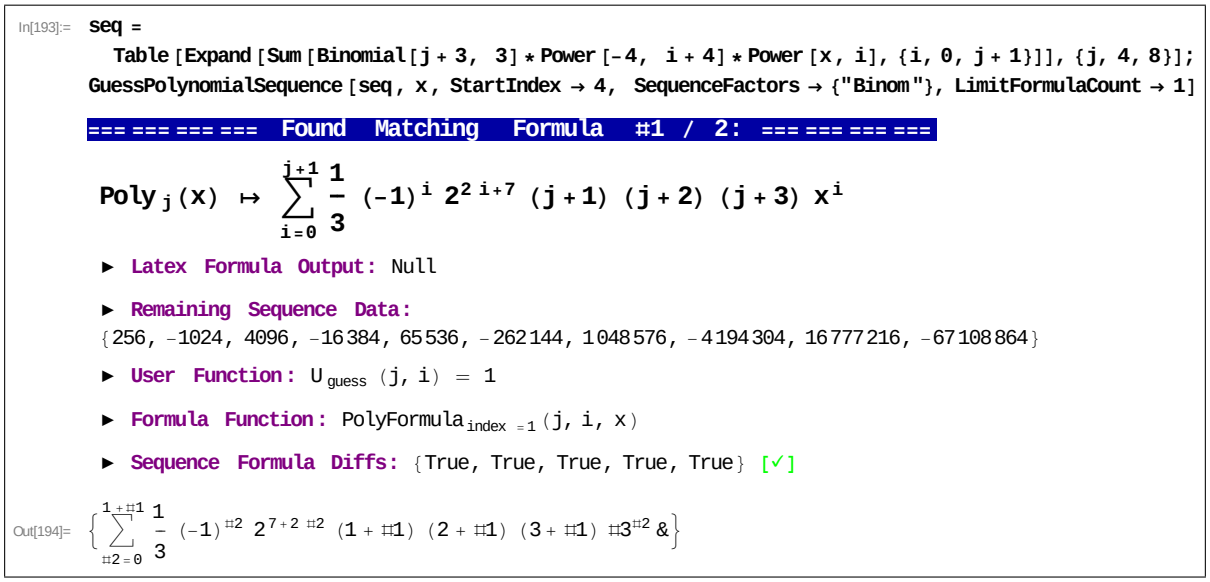

Figure 2.14: Expected Sequence Factors Independent of the Sum Index

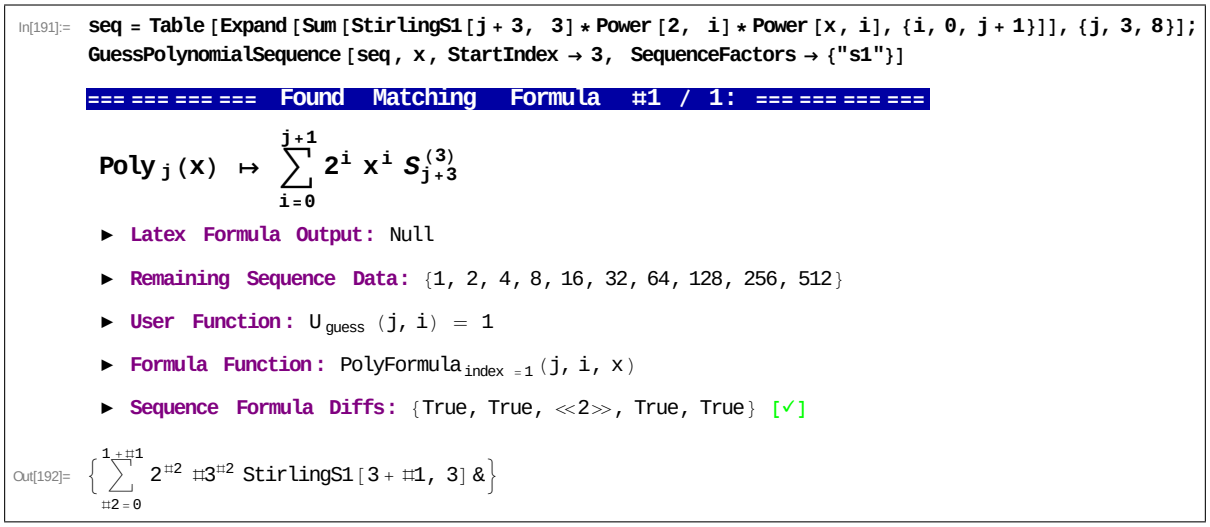

Figure 2.15: Another Example of Expected Sequence Factors Independent of the Sum Index

Other Sequence Formulas

Figure 2.14 and Figure 2.15 cite two additional examples of sequence formulas that the package is able to recognize when the triangular sequence factors expected by the user do not depend on the summation index, i, only the polynomial sequence index, j. In the first example given in Figure 2.14, the expected binomial coefficient factor corresponds to a polynomial in j. In the second example given in Figure 2.15, the expected Stirling number factor corresponds to an expansion in terms of r–order harmonic numbers, $H_{j+2}^{(r)}$, that is reported as the factor of the original Stirling number sequence.

Chapter 3

Implementation of the Package

3.1 Organization of the Package

Before continuing on to the implementation details and other design choices encountered in the *Mathematica* software in the next sections of the thesis, a summary of the rationale for the separation of the package source code files is first given in next this section. In particular, the package source code provided in Appendix A of the thesis is divided into following two separate files:

1. The source code in the file `GuessPolySequenceFormulas.m` listed in Appendix A.2 which implements the core functions and utilities for searching for the sequence formulas returned by the function `GuessPolynomialSequence`; and

2. The source code in the file `GuessSequenceData.m` listed in Appendix A.3 which provides a small suite of useful special triangular sequences that arise in many applications. These out–of–the–box sequence implementations are intended for use as the user–defined expected sequence factors with the package search functions.

Generally speaking, the design choice to separate the triangular sequence data and local *Mathematica* functions to generate these factors should allow the user the flexibility to easily add new coefficient factor sequences to the general formula templates in `GuessPolynomialSequence`. Other than placeholders for the metadata information provided by the example options in sketched in `GuessSequenceData.m`, notice that additional new sequences, or alternate implementations of the built–in triangles in `GuessSequenceData.m` are easily created by simply defining the sequence generator functions, the row index start value and range function, and any other metadata for the sequence desired by the user.

The separation of the file `GuessSequenceData.m` from the core functions and utilities that perform the actual search for formulas in the source code of `GuessPolySequenceFormulas.m` also allows users to easily add new, or plug–and–play with existing, implementations of the expected coefficient factors, employed within the templates local to the package routines. This modular design is intended to allow users to easily

extend and re–implement the triangular sequence forms involved in searching for the sequence formulas without requiring any modification to the core searcg routines in `GuessPolySequenceFormulas.m`. The examples given in Section 2.2 and Section 2.3 are then extended to find other sequence formulas that arise in the context of new applications of interest encountered by users in practice.

3.2 Development of the Package

Package Development in Mathematica

My first choice for the platform for the source code implementation of the package was to use the `Guess` package described in [3]. I settled on writing the implementation of the package in *Mathematica* given its ease of installation compared to the fork of *Axiom* required by the first package, and that I had some existing familiarity with its use. *Mathematica* has several benefits, such as ease of installation on multiple platforms, a large library of built–in functions for performing mathematical operations, including the stock `FindSequenceFunction` function for sequence recognition employed by the package, and also a wide variety of useful existing packages available for use with the system. That being said, learning to write more substantial lines of code and the quirks and conventions of package development with *Mathematica* ended being a substantial investment of time in developing the source code for the thesis package implementation. Some information about coping with testing and debugging of the package code without the use of Wolfram's *Workbench* IDE software is outlined in the implementation notes given in Appendix A.1.

Design Choices and Other Issues Encountered

The following listing provides an explanation of several design choices faced and other issues encountered in the development of the package. Some of the design issues listed below are partially resolved in the current implementation of the package and others are considered in the discussion of future features for the package outlined in Section 3.3 of the thesis below.

- **Polynomials with rational coefficients**: The restriction of the package to processing polynomials with integer coefficients noted in Section 2.1 is due to some of the ambiguities that arise in attempting to determine integer factors present in the forms of strictly rational–valued coefficients. For example, the *Mathematica* function `Divisible` returns `False` to indicate that $\begin{bmatrix}5\\3\end{bmatrix}$ does not divide $\begin{bmatrix}5\\3\end{bmatrix} \cdot \frac{1}{5!}$, even when we seek to identify such a factors of the second expression. Section 3.3 outlines several possible approaches to find formulas for polynomials in $\mathbb{Q}[x]$ based on pre–processing the input polynomials by

a set of transformations produce sequences of modified polynomials in the form of the integer–valued case for use with the package functions.

- **Interpretation of the remaining sequence forms**: The introduction of the polynomial formula templates given in Section 2.1 briefly observed the interpretation of the remaining sequence terms in the polynomial coefficients, say $\mathrm{RS}_1(i) \cdot \mathrm{RS}_2(j-i)$, where at least one of these sequence functions is identically one–valued. This assumption simplifies the search for formulas in these remaining sequence factors, though an example such as

$$\mathrm{RS}_1(i) \cdot \mathrm{RS}_2(j-i) := (-1)^i 4^{j-i} \cdot i!(j+1-i)!$$

  where this is not the case may not produce a matching formula. There does not appear to be an obvious solution to separating the factors in this situation. However, an appropriate choice of the `UserGuessFunction` option to remove additional, non–triangular expected factors in the coefficient formulas can be specified if these remaining factors are easily spotted by inspection of the first several terms of the polynomial sequence.

- **Sequence factors with repeated ones**: One potential optimization to speedup the running time of the package functions for multiple–factor sequence factors is to observe that all of the special triangular sequences provided in the `GuessSequenceData.m` contain one to two one–valued entries in each row of these triangles. In the single–factor formula examples this property of the expected sequences is not significantly noticeable, but finding a more efficient way to process these extra identity factors of every polynomial coefficient should lead to better performance with the double–factor formula examples suggested as applications in Section 2.2.2.

- **Running time limitations with multiple–factor sequence formulas**: Besides the approach to a better handling of the repeated common factors of one suggested above, there are a couple of additional optimization points to consider in future revisions of the package source code. The natural extension of the *Mathematica* code that performed fairly well in testing the implementation of the single–factor sequence examples in the form of (2.1) more or less corresponds to adding an additional layer of nested lists to organize multiple, layered expected factors of the polynomial coefficients. However, this extension did not produce reasonable running times that scale in the computations of the double–factor sequence formulas in (2.2).

  Conventional wisdom on writing efficient code with *Mathematica* also suggests to attempt optimizations

to the existing code using functional programming constructs such as `Map` in place of the `For` loops in the current code for the package. Optimizations to the code using dynamic programming to store sets of previously processed index combinations over individual sequence factors may yield a speedup in the running times for the double–factor and multiple–factor sequence formula cases as well.

3.3 Future Features in the Package

Short Wish List of Features

There is a long laundry list of relatively small improvements to the *Mathematica* source code for the package. These include formatting the triangular sequence factors shown in the program outputs of the figures in Chapter 2 using a `LaTeX`–style `genfrac` display for these coefficients, and implementing many of the runtime option settings starting on line 125 of Appendix A.2. Other larger features I want to add to the program include integrating sequence lookups from the *Online Encyclopedia of Integer Sequences* database and experimenting with multiple packages to use with the `FSFFunction` option in the package.

Processing Polynomial Sequences with Rational Coefficients

One approach to extending the package functionality to recognize formulas for polynomial sequences in $\mathbb{Q}[x]$ is to pre–process the rational–valued coefficients to transform the sequence into the polynomials over the integers already handled by the package routines. Variations of these pre–processing transformations include normalizing the polynomials, its coefficients, or both by exponential factors to clear the denominators of the rational–valued input sequence. For example, let the polynomial $p_j(x) := \sum_i c_i x^i$. Then these transformations are formulated as obtaining the modified polynomials, $\widetilde{p}_j(x)$, as $\widetilde{p}_j(x) := j! \cdot p_j(x)$, as $\widetilde{p}_j(x) := \sum_i i! \cdot c_i x^i$, or in the combined form of $\widetilde{p}_j(x) := \sum_i j! \cdot i! \cdot c_i x^i$, whenever the resulting modified polynomial sequences are in $\mathbb{Z}[x]$.

Another transformation option is applied to rationalize the polynomial sequences, $S_m(n)$, in n defined through the following sums where B_n denotes the (rational) sequence of Bernoulli numbers [2, §6.5]:

$$S_m(n) := \sum_{k=0}^{n-1} k^m = \sum_{k=0}^{m} \binom{m+1}{m-k} \frac{B_{m-k}}{(m+1)} \cdot n^{k+1}.$$

The sequences in the previous equation are normalized by multiplying each polynomial, $S_m(n)$, by the least common multiple of the denominators of each coefficient of n^{k+1} in the formula. Then assuming access to the lookup capabilities of the *Online Encyclopedia of Integer Sequences* database, which contains sequence

entries for both integer sequences of the numerators and denominators of the Bernoulli numbers, obvious factors, of say 691, are recognized to process the full formulas for the sequences of $S_m(n)$ over $\mathbb{Q}[n]$.

Polynomial Expansions With Respect to a Suitable Basis

The discussion given in [5, Appendix A] related to the implementation of the `Rate` package for *Mathematica* states a useful observation that may be adapted to the polynomial formula searches local to this package. Specifically, expressing input polynomial sequences with respect to a "*suitable*" basis, like shifted factorial functions or polynomial terms expressed by binomial coefficients, allows for recognition of sequence formulas that are not apparent in the default expansions of the polynomial sequence variable. Several examples relevant to adapting this idea in the context of the factorization–based approach in this package include the following polynomial sequence expansions [2, Ex. 6.78; §6.2; Ex. 6.68]:

$$\begin{aligned}
\binom{2n}{n}\frac{B_n}{(n+1)} &= \sum_{k=0}^{n}\left\{{n+k \atop k}\right\}\binom{2n}{n+k}\frac{(-1)^k}{(k+1)} \\
\left[{x \atop x-n}\right] &= \sum_{k\geq 0}\left\{{n \atop k}\right\}\binom{x+k}{2n},\ n\geq 0 \\
\left\langle\left\langle {n \atop m}\right\rangle\right\rangle &= \sum_{k=0}^{m}\binom{2n+1}{k}\left\{{n+m+1-k \atop m+1-k}\right\}(-1)^k,\ n>m\geq 0.
\end{aligned}$$

These sequences provide applications related to the polynomial expansions of the Catalan numbers (in n), the Stirling convolution polynomials, $\sigma_n(x)$, and the second–order Eulerian numbers, $\left\langle\left\langle {n \atop m}\right\rangle\right\rangle$, respectively.

Chapter 4

Conclusions

4.1 Concluding Remarks

The package source code listed in Appendix A of the thesis provides a successful "proof of concept" implementation of the logic employed by the approach to the package to recognize polynomial sequence formula types of the noted forms in (2.1) and (2.2). The primary deficiency of the package implementation is current as of this writing is the long running time of the package function `GuessPolynomialSequence` when processing double–factor and multiple–factor sequence formulas of the form outlined in (1.8). Single–factor polynomial sequence formulas in the form of (2.1) like those cited in (1.1) of the introduction are already somewhat easy, though not trivial, to guess by the user. For the package to be really useful in practice, the sequence recognition routines provided through the wrapper function `GuessPolynomialSequence` should be able to guess double–factor formulas of the form in (2.2) fairly quickly and efficiently out–of–the–box.

The examples given in Chapter 2 provide several non–trivial uses of the package for recognizing single–factor polynomial formulas of the first sequence form in (2.1). These and related applications corresponding to polynomials that satisfy a single–factor formula of this variety are easily and fairly quickly recognized by the package given an accurate user–defined setting of the `SequenceFactors` runtime option to `GuessPolynomialSequence`. Many other formulas for related single–factor polynomial sequences are also easily obtained using the package and by straightforward extensions of the sequence data package file, `GuessSequenceData.m`, listed in Appendix A.3.

For polynomial sequences that satisfy a double–factor formula of the second form in (2.2), and more generally a multiple–factor formula in the form stated in (1.8) where $r \geq 3$, the current package implementation is unable to quickly search for matching formulas without a somewhat manual limited setting of the `IndexOffsetPairs` option provided at runtime. The sample output for the examples given in Figure 2.5 and Figure 2.6 show the usage of the package for handling double–factor sequence formulas with an appropriate setting of this option. In future revisions of the package, it should ideally be possible for the package to quickly obtain formulas for these sequence cases without the user manually resetting the default

search options used with the `GuessPolynomialSequence` function provided by the package.

4.2 Future Research Topics

The next sections discuss several topics for future research suggested by the implementation of the software package for the thesis. These future research topics include a new variation of integer factorization algorithms motivated by the factorization–based approach to handling the user–defined expected sequence factors in the package routines, as well as additional topics for future exploration to extend the current capabilities of the univariate polynomial sequence recognition in the package. The extension of the current package functionality to recognizing polynomials in a single variable with rational–valued coefficients is already considered in Section 3.3 of the thesis above.

Sequence–Based Integer Factorization Algorithms

The treatment of the user–defined expected sequence factors as "primitives" in the formulas returned by the package functions motivates the construction of a class of integer factorization algorithms formulated briefly in the discussion below. Much like computing the prime factorization of an arbitrary integer, this class of algorithms should compute the decomposition of an integer into a product of elements over some specified set of integer sequences where the elements of these sequences are treated as "atoms" in the factorization returned by the procedure.

Stated more precisely: given an integer i (or some set of integer–valued polynomial coefficients) and a list of k integer sequences, $\{S_1, S_2, \ldots, S_k\}$, we seek the most efficient way to decompose the integer into all possible products of integer factors of the form

$$i := f_1 \cdot f_2 \cdot \cdots \cdot f_k \times r, \tag{4.1}$$

where the factor f_i belongs to the sequence S_i (for each $1 \leq i \leq k$), and where the remaining factor term, r, is reserved for later processing. The computation of the list of all factors of the form in (4.1) can be computed over some specified number of elements of each sequence, or a fixed number of rows for the case of a triangular sequence, S_i. It seems reasonable to expect that such an algorithms must employ the prime factorizations of the individual factor sequences, F_i. We also seek a solution in the general case, though of course it may be possible to derive sequence–specific procedures, say, to recognize factors of the Stirling number or binomial coefficient triangles.

The need for this type of factorization is apparently new, as searches for such subroutines to employ

within the package returned no useful known results, though it is possible that there are existing prime factorization algorithms that may be especially well–suited, or adapted, to this purpose. This required factorization procedure is handled as an inefficient implementation of an oracle of sorts within the current implementation of the package.

Multiple Sum Formulas and Polynomial Sequences in Multiple Variables

The package functionality does not currently consider polynomial sequence formulas involving multiple sums or sequences with more than one variable. Finding the terms in the complete formula given by the double sum in (1.5) involving the polynomials in (1.7) is one example of the utility of this type of extension to the package. The next examples cite other specific applications of extending the package functionality to process polynomials and polynomial formulas of these particular types.

Example I: A Multiple Sum Identity for the Function $S_m(n)$

Another example of this type of formula is related to the example for the function, $S_m(n) := \sum_{0 \leq k < n} k^m$, defined as the sum cited as an application in Section 3.3. This function is expanded as a polynomial in n through the single–summation formula given in with rational coefficients involving the Bernoulli numbers, B_n. This sum also satisfies the form of a double sum involving the Stirling number triangles expanded as follows [2, §6.5]:

$$S_m(n) = \sum_{j=0}^{m} \sum_{k=0}^{j+1} \left\{ {m \atop j} \right\} \left[{j+1 \atop k} \right] \cdot \frac{(-1)^{j+1-k}}{(j+1)} \cdot n^k.$$

The sum in the previous equation provides an application of a polynomial sequence formula in the single variable, n, whose formula in this case is specified in terms of a multiple, nested summation over the index inputs to the Stirling number factors.

Example II: Formulas for Derivatives of Lambert Series Expansions

Consider the following double summation formula involving both Stirling number triangles for the j^{th} derivatives with respect to q of terms that arise in the *Lambert series* expansions related to many special arithmetic functions:

$$q^j \cdot \frac{d^{(j)}}{dq^{(j)}} \left[\frac{q^n}{(1-q^n)} \right] = \sum_{m=0}^{j} \sum_{i=0}^{m} \left[{j \atop m} \right] \left\{ {m \atop i} \right\} (-1)^{j+1} (-1)^{i+1} \cdot i! \cdot \frac{n^m q^n}{(1-q^n)^{i+1}}. \tag{4.2}$$

Notice that the formula in (4.2) may be treated as a polynomial in powers of the two variables, n and $(1-q^n)^{-1}$. A possible extension to the current package functionality is to construct sequence recognition

routines to recognize double sum formulas, including polynomial sequence formulas in more than one variable, such as the nested multiple sum formulas given as the expansion of (4.2).

One application of finding formulas for the particular polynomial expansions in (4.2) is related to the Lambert series for the *sum of divisors function*, $\sigma_\alpha(m)$, defined as

$$\sigma_\alpha(n) = \sum_{d|n} d^\alpha \qquad \text{where} \qquad \sum_{n=1}^{\infty} \frac{\sigma_\alpha(n)}{n^s} = \zeta(s)\zeta(s-\alpha)$$

for each $n \geq 1$, fixed $\alpha, s \in \mathbb{C}$ such that $\Re(s) > \max(1, 1+\Re\alpha)$, and where $\zeta(s)$ denotes the *Riemann zeta function* [6, §27.7, §27.4]. Then the stated formulas for these expansions of the j^{th} derivatives with respect to q given in (4.2) can, for example, be used to establish the Dirichlet–series–related generating functions over the sequences of $\sigma_\alpha(n)$ given by the following series:

$$\begin{aligned}
\sum_{n=1}^{\infty} \frac{\sigma_\alpha(n)}{n} q^n &= \sum_{j=1}^{\infty} \frac{(-1)^{j-1} \cdot q^j}{j!} \left(H_j\right) \times \frac{d^{(j)}}{dq^{(j)}} \left[\sum_{n=1}^{\infty} \frac{n^\alpha \cdot q^n}{(1-q^n)}\right] \\
\sum_{n=1}^{\infty} \frac{\sigma_\alpha(n)}{n^2} q^n &= \sum_{j=1}^{\infty} \frac{(-1)^{j-1} \cdot q^j}{2 \cdot j!} \left(H_j^2 + H_j^{(2)}\right) \times \frac{d^{(j)}}{dq^{(j)}} \left[\sum_{n=1}^{\infty} \frac{n^\alpha \cdot q^n}{(1-q^n)}\right] \\
\sum_{n=1}^{\infty} \frac{\sigma_\alpha(n)}{n^3} q^n &= \sum_{j=1}^{\infty} \frac{(-1)^{j-1} \cdot q^j}{6 \cdot j!} \left(H_j^3 + 3H_j H_j^{(2)} + 2H_j^{(3)}\right) \times \frac{d^{(j)}}{dq^{(j)}} \left[\sum_{n=1}^{\infty} \frac{n^\alpha \cdot q^n}{(1-q^n)}\right].
\end{aligned}$$

Notice that many other similar double and triple summation identities related to the expansions in (4.2). These summations provide additional applications of the extension of the package to handling multiple sums of these types.

References

[1] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*, Cambridge University Press, 2009.

[2] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, 1994.

[3] W. Hebisch and M. Rubey, Extended rate, more GFUN, *Journal of Symbolic Computation* **46** (2011), 889–903. See `http://axiom-wiki.newsynthesis.org/GuessingFormulasForSequences`.

[4] M. Kauers, Summation algorithms for Stirling number identities, *Journal of Symbolic Computation* **42** (2007), 948–970. See `http://www.risc.jku.at/research/combinat/software/ergosum/RISC/Stirling.html`.

[5] C. Krattenthaler, Advanced determinant calculus, *Séminaire Lotharingien de Combinatoire* **42** (1999), 1–67. See `http://www.mat.univie.ac.at/~kratt/rate/rate.html`.

[6] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, eds., *NIST Handbook of Mathematical Functions*, Cambridge University Press, 2010.

[7] M. Petkovsek, H. S. Wilf, and D. Zeilberger, $A = B$, A K Peters, Ltd., 1996.

[8] B. Salvy and P. Zimmermann, Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable, *ACM Transactions on Mathematical Software* **20** (1994), 163–177.

[9] N. J. A. Sloane, The Online Encyclopedia of Integer Sequences, `http://oeis.org`.

Appendix A

Source Code

A.1 Implementation Notes

The package source code current as of April 28, 2014 is archived in the Appendix A.2 and Appendix A.3 sections of this Master's thesis deposited in May 2014 at the University of Illinois at Urbana–Champaign. The rationale for the organization of the package into the two separate source code files is explained in more detail by Section 3.1 of the thesis above. It is intended that the second file, `GuessSequenceData.m`, that implements the current triangular sequence functions and metadata information used for searches with the function `GuessPolynomialSequence` be extended to include implementations of sequences relevant to finding formulas for polynomials in applications other than those strictly provided by the examples of Chapter 2 in this thesis.

A.1.1 Programming in Mathematica

The source code for the package in Appendix A.2 and Appendix A.3 contains a fair number of inline comments and should be accessible to users familiar with programming in *Mathematica*. An additional reference explaining the conventions of package development is available in the standard documentation within *Mathematica*[1]. The book *Mathematica Programming: An Advanced Introduction* by Leonid Shifrin also contains general information about functional programming and other conventions in *Mathematica* and is available freely online in multiple formats.

Conventions and Programming Style

The source code for both package files is mostly consistent with the *Mathematica* programming conventions outlined in the standard documentation references[2]. The package code uses Java–style capitalization in variable names since *Mathematica* variables cannot contain underscore characters. A local convention employed

[1] See the following links in the built–in *Mathematica* documentation: `tutorial/ModularityAndTheNamingOfThingsOverview`.

[2] See the following links in the built–in *Mathematica* documentation: `tutorial/SomeGeneralNotationsAndConventions`, `tutorial/NamingConventions`, and `tutorial/GettingUsedToMathematica`.

within the package is that most non–public, internal begin with a lowercase letter (in contrast to the naming conventions for the standard functions built–in to *Mathematica*).

Other Programming Notes

One particular issue to notice is that *Mathematica*'s handling of nested scopes and private package variables requires a somewhat non–intuitive use of the scoping construct `With` briefly explained in the *Possible Issues* section of the documentation, `ref/With`, for the function. This usage of the `With` construct is found starting on lines 601 and 1233 in the source for the file `GuessPolySequenceFormulas.m` in Appendix A.2.

A.1.2 Testing and Debugging

The testing and debugging of the package was performed without the use of Wolfram's *Workbench* IDE. The *Workbench* software includes a debugger, profiler, and unit testing capabilities for use with *Mathematica* code. Tests for the package may be coded as shell scripts using the `MathematicaScript -script` command shipped with *Mathematica*. The package defines the function `DiffSequenceFormula` which may be used to determine whether the formulas returned by `GuessPolynomialSequence` correctly match the polynomials passed as input to the function. This approach can be used to script a test suite or unit tests without the need for the *Workbench* software.

There are also several modified conditional print functions defined within the source code in the file `GuessPolySequenceFormulas.m` for debugging purposes, including the functions `PrintDebug`, `PrintStatus`, `PrintMatch`, `PrintMatchInfo`, and `PrintVariable`. Like the stock *Mathematica* `Print` and `PrintTemporary` functions, all of these functions take a variable number of arguments, except for `PrintVariable` which only takes a single argument as input. The messages printed by these functions placed in the source code are enabled by setting the option `EnableDebugging->True` passed to the function `GuessPolynomialSequence` at runtime. Note that enabling the printing of all of these debugging messages will generate a typically huge number of messages in calling `GuessPolynomialSequence`. The conditional printing of these messages is also configured manually using the private package functions `EnableDebugging[]` and `DisableDebugging[]`. The local package function `ConfigDebuggingMessages` is used to enable the selective printing of only certain classes of debugging messages.

A.2 GuessPolySequenceFormulas.m

Mathematica Source Code

```
(*************************************************************************)
(*************************************************************************)
(******************** : GuessPolySequenceFormulas.m: ********************)
(*************************************************************************)
(*************************************************************************)

BeginPackage["GuessPolySequenceFormulas`", {"GuessSequenceData`"}]

(*************************************************************************)
(**** : Package revision metadata information:                       ****)
(*************************************************************************)

PackageName = "GuessPolySequenceFormulas.m";
PackageVersion = "2014.04.28-v5";
PackageAuthor = "Maxie D. Schmidt";
PackageShortDesc =
    "Package routines and utilities for intelligent guessing of " <> "\n" <>
    "univariate polynomial sequence formulas.";

(*************************************************************************)
(**** : Create default usage information for the package functions:   ****)
(**** : Provide detailed usage information for the package functions: ****)
(*************************************************************************)

GuessPolynomialSequence::usage =
    "GuessPolynomialSequence[polyseq, polyVar]";

DiffSequenceFormula::usage =
    "DiffSequenceFormula[polySeq, polyVar, polyFormula, StartIndex -> 1]";

(*************************************************************************)
(**** : Package error handling and messages:                         ****)
(*************************************************************************)

GPSFPkgMsgs::Errors::NonPolynomialInput =
    "The input `1` is not a polynomial in `2`";

GPSFPkgMsgs::Warnings::InsufficientSeqElements =
    "Only `1` sequence elements passed as input, " <>
    "possibly an insufficient number to guess formulas " <>
    "(consider passing \[GreaterEqual] `2` polynomials as input)";
GPSFPkgMsgs::Warnings::InsufficientFactorData =
    "Only `1` rows of coefficient factor data specified, " <>
    "possibly an insufficient number to correctly process formulas " <>
    "(consider setting TriangularSequenceNumRows" <>
    "\[LongRightArrow]`2` or more)";

GPSFPkgMsgs::BuildLocalSequenceData::SeqFactorsList =
    "Invalid list of sequence factors `1`";
GPSFPkgMsgs::BuildLocalSequenceData::InvalidSeqID =
    "Invalid sequence ID `1`";
GPSFPkgMsgs::PackageMultipleTSeqData::ListLength =
    "Upper and Lower list lengths do not match (`1` / `2`)";
GPSFPkgMsgs::GuessMultipleFactorPolySequence::PolySeqDegree =
    "Unable to compute the polynomial sequence degrees";

(*************************************************************************)
(**** : Package configuration and global settings: ****)
(*************************************************************************)

Begin["`PkgConfig`"]

Debugging = True;
KeepStatusMessages = True;
KeepStatusMatchMessages = True;
AllowSymbolicData = False;

```

```
End[] (* PkgConfig *)

Begin["`Private`"]

(*****************************************************************)
(**** : Misc utilities for the package: ****)
(*****************************************************************)

PrintDebug[args___] := Module[{debugFlag, printPrefix},
    debugFlag = GuessPolySequenceFormulas`PkgConfig`Debugging;
    If[debugFlag,
        printPrefix = Style["DEBUGGING: ", Red, Bold];
        Print[printPrefix, args];
    ];
]

PrintStatus[msgs___] := Module[{keepMsgFlag, printPrefix},
    keepMsgFlag = GuessPolySequenceFormulas`PkgConfig`KeepStatusMessages;
    If[keepMsgFlag,
        printPrefix = Style["STATUS: ", Blue, Bold];
        Print[printPrefix, msgs];
    ];
]

PrintMatch[msgs___] := Module[{keepMsgFlag, printPrefix},
    keepMsgFlag = GuessPolySequenceFormulas`PkgConfig`KeepStatusMatchMessages;
    If[keepMsgFlag,
        printPrefix = Style["MATCH_DATA: ", Cyan, Bold, Underlined];
        Print[printPrefix, msgs];
    ];
]

PrintMatchInfo[msgs___] := Module[{keepMsgFlag, printPrefix},
    keepMsgFlag = GuessPolySequenceFormulas`PkgConfig`KeepStatusMatchMessages;
    If[keepMsgFlag,
        printPrefix = Style["MATCH_INFO: ", Green, Bold, Underlined];
        Print[printPrefix, msgs];
    ];
]

PrintVariable[var_] := Module[{keepMsgFlag, printPrefix},
    keepMsgFlag = GuessPolySequenceFormulas`PkgConfig`KeepStatusMatchMessages;
    If[keepMsgFlag,
        printPrefix = Style["VAR_VALUE: ", Orange, Bold, Underlined];
        Print[printPrefix, SymbolName[var], ": ", var];
    ];
]

ConfigDebuggingMessages[setOnOff_:{False,False,False}] := Block[{},
    GuessPolySequenceFormulas`PkgConfig`Debugging = setOnOff[[1]];
    GuessPolySequenceFormulas`PkgConfig`KeepStatusMessages = setOnOff[[2]];
    GuessPolySequenceFormulas`PkgConfig`KeepStatusMatchMessages = setOnOff[[3]];
];

EnableDebugging[] := ConfigDebuggingMessages[ConstantArray[True, 3]];
DisableDebugging[] := ConfigDebuggingMessages[ConstantArray[False, 3]];


DefaultPkgConfigOptions = {
    StartIndex -> 1,
    IndexMultiples -> {0, 1},
    IndexOffsetPairs -> Null,
    SequenceFactors -> {"S1"},      (* See GuessSequenceData.m *)
    LimitFormulaCount -> Infinity, (* Limit the number of returned formulas *)
    ClearLocalSequenceData -> True,
    FactorFunction -> Null,         (* Null for local handling functions, or
                                       possibility for a user-defined function *)
```

```
135         UserGuessFunction -> DefaultUserGuessFn,
136         FSFFunction -> FindSequenceFunction,
137         FSFFnExpectedStartIndex -> 1,
138         FSFImposeTimingConstraints -> False,
139         FSFTimingConstraintSeconds -> 1.0,
140         CheckForConstantSequences -> True,
141         RequireMatchingULIndexFormula -> True,
142         RequireMatchingRemSequenceFormula -> False,
143         IssueAllWarnings -> False,
144         AllowSymbolicData -> False,
145         ProcessRemSequenceFormulas -> True,
146         ReverseRemSeqFormulaIndex -> False,
147         TriangularSequenceNumRows -> 12,
148         ExtractSequenceDataFunction -> DefaultExtractSequenceDataFn,
149         ExtractUIIndexDataFunction -> DefaultExtractUIIndexDataFn,
150         ReturnNullSequenceType -> Null,
151         RemSeqProcFnFlag -> Null,
152         GenerateNotebookOutput -> True,
153         GeneratePlaintextOnly -> False,
154         ReturnFormulasOnly -> True,
155         ReturnFullFormulaData -> False,
156         DisplayVars -> {j, i},
157         PrintFormulas -> True,
158         PrintLaTeXFormulas -> True,
159         PrintPkgDebuggingMsgs -> {True, True, Print}, (* {T/F, VerboseQ, PrintFn} *)
160         PrintPkgStatusMsgs -> {True, True, Print},    (* {T/F, VerboseQ, PrintFn} *)
161         PrintPkgMatchMsgs -> {True, True, Print},     (* {T/F, VerboseQ, PrintFn} *)
162         PrintRuntimeTimingMsgs -> {True, False, PrintTemporary},
163         SimplifyFormulas -> True,
164         SimplifyFns -> {Simplify},
165         FullSimplifyFormulas -> False,
166         FullSimplifyFns -> {PowerExpand, FunctionExpand, FullSimplify},
167         DiffPolySequenceFormulas -> True,
168         EnableDebugging -> False,
169         AbortOnSymmetricRemSequence -> True,
170         RequireSeqDegreeFormula -> True, (* Otherwise, specify C[i][#1] *)
171         ProcessPolyDataFn -> Null
172 };
173
174 GetDefaultConfigOption[option_] := (option /. DefaultPkgConfigOptions);
175
176 (*************************************************************************)
177 (**** : Configuration of (semi) static data:                         ****)
178 (*************************************************************************)
179
180 PkgNumSequenceFactors = 0;
181 PkgSequenceData = {};
182
183 GetTriangularSequenceData[numRows_, startRowIndex_,
184                           seqGenFn_, rowRangeFn_] :=
185 Module[{seqData, seqPkgFn, row, rowIndex, colIndexRange, nextRow},
186
187       seqData = {};
188       seqPkgFn = PackageTriangularSequenceData[#1, #2, seqGenFn[#1, #2]]&;
189       For[row = 0, row < numRows, row++,
190             rowIndex = row + startRowIndex;
191             colIndexRange = Range[##]& @@ rowRangeFn[rowIndex];
192             nextRow = Map[seqPkgFn[rowIndex, #1]&, colIndexRange];
193             seqData = Union[seqData, nextRow];
194       ];
195
196       PrintDebug["GetTSeqData seqData: ", seqData];
197       Return[seqData];
198
199 ]
200
201 Options[BuildLocalSequenceData] = DefaultPkgConfigOptions;
```

```
BuildLocalSequenceData[cfgOptions : OptionsPattern[]] :=
Module[{sequenceIDs, numRows, sindex, seqID, seqMetaData, startRowIndex,
        seqGenFn, rowRangeFn, curSeqData},

      sequenceIDs = OptionValue[SequenceFactors];
      If[(Head[sequenceIDs] != List) || (Length[sequenceIDs] == 0),
            Message[GPSFPkgMsgs::BuildLocalSequenceData::SeqFactorsList,
                  sequenceIDs];
            Return[Null];
      ];

      (** : Reset the local sequence data storage array: **)
      PkgNumSequenceFactors = Length[sequenceIDs];
      PkgSequenceData = {};

      numRows = OptionValue[TriangularSequenceNumRows];
      For[sindex = 1, sindex <= Length[sequenceIDs], sindex++,

            (** : Uses the local sequences package data / implementations: **)
            seqID = sequenceIDs[[sindex]];
            seqMetaData = QuerySequenceMetaData[seqID];
            If[Length[seqMetaData] == 0,
                  Message[GPSFPkgMsgs::BuildLocalSequenceData::InvalidSeqID,
                        seqID];
                  PkgSequenceData = {};
                  PkgNumSequenceFactors = 0;
                  Return[False];
            ];
            PrintDebug[seqKeys, " ", seqKey, " ", seqMetaData];

            startRowIndex = (StartRowIndex) /. seqMetaData;
            seqGenFn = (GeneratorFunction) /. seqMetaData;
            rowRangeFn = (RowRangeFunction) /. seqMetaData;

            curSeqData = GetTriangularSequenceData[numRows, startRowIndex,
                                                   seqGenFn, rowRangeFn];
            PkgSequenceData = Append[PkgSequenceData, curSeqData];

      ];

      Return[True];

]

FactorIntegerBySequences[int_] :=
Module[{firstSeqFactorData, numSequences, fullFactorData, sindex, curSeqData,
        nextFullFactorData, eindex, prevFactors, prevInt, divCondFn,
        curSeqFactorData, nextFactorDataFn, nextFactorData},

      firstSeqFactorData = FactorIntegerBySequence[int];
      numSequences = PkgNumSequenceFactors;

      (** : Add brackets around the first sequence index data: **)
      (*fullFactorData = Map[{ {ExtractTSeqULIndexData[#1]},
                              ExtractTSeqValue[#1] }&, firstSeqFactorData]; *)
      fullFactorData = firstSeqFactorData;

      For[sindex = 2, sindex <= numSequences, sindex++,

            curSeqData = PkgSequenceData[[sindex]];
            nextFullFactorData = {};
            For[eindex = 1, eindex <= Length[fullFactorData], eindex++,

                  prevFactors = ExtractTSeqULIndexData[ fullFactorData[[eindex]] ];
                  prevInt = ExtractTSeqValue[ fullFactorData[[eindex]] ];
                  divCondFn = Divisible[prevInt, ExtractTSeqValue[#1]]&;
                  curSeqFactorData = Select[curSeqData, divCondFn];
```

```
                localReplaceFn = ReplacePart[#1,
                                 2 -> (prevInt / ExtractTSeqValue[#1])]&;
                curSeqFactorData = Map[localReplaceFn, curSeqFactorData];

                nextFactorDataFn =
                     { Append[prevFactors, ExtractTSeqULIndexData[#1]],
                       ExtractTSeqValue[#1] }&;
                nextFactorData = Map[nextFactorDataFn, curSeqFactorData];
                nextFullFactorData = Union[nextFullFactorData,
                                           nextFactorData];

          ];
          fullFactorData = nextFullFactorData;

     ];

     Return[fullFactorData];

]

FactorIntegerBySequence[int_] :=
Module[{seqData, leadingIntCoeff, allowSymbolicData, divCondFn,
        factorData, replaceFn, rFactorData},

     seqData = PkgSequenceData[[1]];

     (** : Handle symbolic data by extracting the integer coefficient: **)
     leadingIntCoeff = int;
     allowSymbolicData = GuessPolySequenceFormulas`PkgConfig`AllowSymbolicData;
     If[allowSymbolicData && !IntegerQ[int] &&
        (Length[int] > 0) && (Head[int[[1]]] == Integer),
          leadingIntCoeff = int[[1]];
     ];

     (** : First determine which sequence elements divide the integer: **)
     divCondFn = Divisible[leadingIntCoeff, ExtractTSeqValue[#1]]&;
     factorData = Select[seqData, divCondFn];

     (** : Then replace the sequence elements with the remainder of int: **)
     replaceFn = ReplacePart[#1, 2 -> (int / ExtractTSeqValue[#1])]&;

     replaceFn = {{ExtractTSeqULIndexData[#1]}, int / ExtractTSeqValue[#1]}&;
     rFactorData = Map[replaceFn, factorData];
     Return[rFactorData];

]

(**** : Helper routines for handling triangular sequence data: ****)
PackageTriangularSequenceData[upperi_, loweri_, value_] :=
Module[{indexData, valueData, rData},

     indexData = {upperi, loweri};
     valueData = value;
     rData = {indexData, valueData};
     Return[rData];

]

ExtractTSeqULIndexData[edata_] := Module[{},
     Return[ edata[[1]] ];
]

ExtractTSeqUpperIndex[edata_] := Module[{},
     Return[ edata[[1]][[1]] ];
]

ExtractTSeqLowerIndex[edata_] := Module[{},
```

```
336       Return[ edata[[1]][[2]] ];
337   ]
338
339   ExtractTSeqValue[edata_] := Module[{},
340       Return[ edata[[2]] ];
341   ]
342
343   PackageMultipleTSeqData[uiList_, liList_, remValue_] :=
344   Module[{ulIndexData, rData},
345
346       If[Length[uiList] != Length[liList],
347           Message[GPSFPkgMsgs::PackageMultipleTSeqData::ListLength,
348                   uiList, liList];
349           Return[{ {}, remValue}];
350           Return[Null];
351       ];
352
353       If[Length[uiList] == 0, (* Return empty list for the sequence data *)
354           Return[{ {}, remValue}];
355       ];
356
357       ulIndexData = MapIndexed[{uiList[[ First[#2] ]], liList[[ First[#2] ]]}&,
358                                Range[Length[uiList]]];
359
360       rData = {ulIndexData, remValue};
361
362       Return[rData];
363
364   ]
365
366   ExtractMultipleTSeqUpperIndices[edata_] :=
367   Module[{ulIndexData, uIndexList},
368
369       ulIndexData = ExtractTSeqULIndexData[edata];
370       uIndexList = Map[(#1)[[1]]&, ulIndexData];
371       Return[uIndexList];
372
373   ]
374
375   ExtractMultipleTSeqLowerIndices[edata_] :=
376   Module[{ulIndexData, lIndexList},
377
378       ulIndexData = ExtractTSeqULIndexData[edata];
379       lIndexList = Map[(#1)[[2]]&, ulIndexData];
380       Return[lIndexList];
381
382   ]
383
384   ExtractRemainingValue[edata_] := ExtractTSeqValue[edata]
385
386   DefaultUserGuessFn[polyIndex_, sumIndex_] := 1;
387
388   IsEmptySet[set_] := (Length[set] == 0);
389   IsConstantSequence[seq_] := Module[{},
390       Return[!IsEmptySet[seq] && (Length[Tally[seq]] == 1)];
391   ]
392
393   ConstantFn[constant_, args___] := constant;
394   GetConstantFn[constant_] := ConstantFn[constant, ##]&
395   GetNullValuedFn[] := GetConstantFn[Null]
396   SetAttributes[ConstantFn, Constant];
397
398   GetFixedSequenceDataFn[fixedSeqData_, startIndex_:0,
399                          genParamIndex_:0] := Which[
400       #1 < startIndex, SEQ[genParamIndex][#1],
401       #1 >= startIndex + Length[fixedSeqData], SEQ[genParamIndex][#1],
402       _, fixedSeqData[[1 - startIndex + #1]]
```

```
];

PackagePolyIndexMatchData[uiIndexData_, seqData_] := {uiIndexData, seqData};
DefaultExtractUIIndexDataFn[elem_] := elem[[1]];
DefaultExtractSequenceDataFn[elem_] := elem[[2]];

Options[DefaultPackageFormulaDataFn] = DefaultPkgConfigOptions;
DefaultPackageFormulaDataFn[ulIndexFnData_, remSeqFnData_,
                            cfgOpts : OptionsPattern[]] :=
Module[{},

    Return[Null];

];

Options[DiffSequenceFormula] = DefaultPkgConfigOptions;
DiffSequenceFormula[polySeq_, polyVar_, polyFormula_,
                    cfgOpts : OptionsPattern[]] :=
Module[{seqStartIndex, indexAdjust, seqDiff, diffFn, polyDiffs},

    seqStartIndex = OptionValue[StartIndex];
    indexAdjust = seqStartIndex - 1;
    si = Unique[si];
    seqDiff = (#1 - polyFormula[#2 + indexAdjust, si, polyVar])&;
    diffFn = seqDiff[#1, #2]&;
    polyDiffs = MapIndexed[(Simplify[Expand[diffFn[#1, First[#2]]]] === 0)&,
                           polySeq];
    Return[polyDiffs];

]

(**** : Begin ComputeSequenceFormula[...] helper functions : ****)

(*formatReturnDataFn[matchQ_, seqFn_] := {matchQ, seqFn}; *)
formatReturnDataFn[matchQ_, seqFn_] := Module[{},
    Return[{matchQ, seqFn}];
]

(**** : End ComputeSequenceFormula[...] helper functions : ****)

Options[ComputeSequenceFormula] = DefaultPkgConfigOptions;
ComputeSequenceFormula[seqData_, startIndex_,
                       cfgOpts : OptionsPattern[]] :=
Module[{FSFFn, expectedStartIndex, checkConstantSequences, initFormula,
        constFn, indexAdjust, seqFormulaFn, simplifyQ, fullSimplifyQ,
        simplifyFns, simpFormula, simpSeqFormulaFn},

    FSFFn = OptionValue[FSFFunction];
    expectedStartIndex = OptionValue[FSFFnExpectedStartIndex];
    checkConstantSequences = OptionValue[CheckForConstantSequences];

    PrintDebug["seqData: ", seqData];
    PrintDebug["startIndex: ", startIndex];
    PrintDebug["CheckConst = ", checkConstantSequences, "; expected = ",
               expectedStartIndex];

    (** : attempt to compute a closed-form formula for the sequence: **)
    initFormula = FSFFn[seqData];

    (** : Return NULL if FSFFn[...] cannot find a formula expression: **)
    If[Head[initFormula] == FSFFn,
        If[checkConstantSequences && IsConstantSequence[seqData],
            constFn = GetConstantFn[ seqData[[1]] ];
            Return[formatReturnDataFn[True, constFn]];
        ];
        Return[formatReturnDataFn[False, GetConstFn[Null]]];
    ];
```

```
470
471         indexAdjust = startIndex - expectedStartIndex;
472         PrintDebug["indexAdjust: ", indexAdjust];
473
474         seqFormulaFn = initFormula /. (# -> # - indexAdjust);
475
476         (** : Attempt to simplify the formula if the options are set : **)
477         (** : Note: FullSimplifyFormulas takes precedence over the   : **)
478         (** :       default simplify option SimplifyFormulas         : **)
479         simplifyQ = OptionValue[SimplifyFormulas];
480         fullSimplifyQ = OptionValue[FullSimplifyFormulas];
481         simplifyFns = Which[fullSimplifyQ, OptionValue[FullSimplifyFns],
482                             simplifyQ, OptionValue[SimplifyFns], _, {}];
483
484         If[(!simplifyQ && !fullSimplifyQ) || IsEmptySet[simplifyFns],
485             Return[formatReturnDataFn[True, seqFormulaFn]];
486         ];
487
488         (** : Apply the simplification functions in default order: **)
489         PrintStatus["Pre seqFormulaFn: ", seqFormulaFn];
490         xi = Unique[xvar];
491         simpFormula = Last[ComposeList[simplifyFns, seqFormulaFn[xi]]];
492         simpSeqFormulaFn = Evaluate[simpFormula /. (xi -> #1)]&;
493         PrintStatus["Post seqFormulaFn: ", simpSeqFormulaFn];
494
495         Return[formatReturnDataFn[True, simpSeqFormulaFn]];
496
497 ]
498
499 (**** : Perform pre-processing of the input polynomial sequence terms : ****)
500 PreProcessPolynomialData[inputPolyForm_, polyIndex_, polyVar_, userGuessFn_,
501                          cfgOpts : OptionsPattern[],
502                          processOpts : OptionsPattern[]] :=
503 Module[{pxvar, cfList, processCoeffFn, modcfList, modPolySum, modPoly,
504         modTerms, modPolyMetadata, rData},
505
506         (** : Perform basic error checking : **)
507         PrintStatus["In pre-process ... PolyIndex = ", polyIndex];
508
509         (** : Adjust / scale the original coefficients: **)
510         cfList = CoefficientList[inputPolyForm, polyVar];
511         processCoeffFn = Cancel[(#1) / userGuessFn[polyIndex, #2]]&;
512         modcfList = MapIndexed[processCoeffFn[#1, First[#2] - 1]&, cfList];
513
514         (** : Get the new, modified polynomial function : **)
515         modPolySum = MapIndexed[(#1) * Power[pxvar, First[#2] - 1]&, modcfList];
516         modPoly = Function[pxvar, modPolySum][polyVar];
517         modTerms = MapIndexed[(#1) * Power[polyVar, First[#2] - 1]&, modcfList];
518         modPoly = Plus @@ modTerms;
519         modPolyMetadata = Null;
520
521         (** : Setup the returned polynomial / accounting information: **)
522         rData = {ModPolyFn -> modPoly,
523                  ModPolyMetadata -> modPolyMetadata};
524         Return[rData];
525
526 ]
527
528 Options[ProcessPolynomialWrapper] = DefaultPkgConfigOptions;
529 ProcessPolynomialWrapper[poly_, polyIndex_, polyVar_, indexOffsets_,
530                          factorFn_, cfg : OptionsPattern[]] :=
531         Module[{cflist, cfFactorData, processPolyDataFn, matchData},
532
533         (** : Perform error checking: **)
534         PrintStatus["In Process Fn ... PolyIndex = ", polyIndex, " p=", poly, " v=",
        polyVar];
535
```

```
536       (** : Get the coefficient factor data (apply user guess function): **)
537       cflist = CoefficientList[poly, polyVar];
538       cfFactorData = Map[factorFn, CoefficientList[poly, polyVar]];
539
540       (** : Compute the possible matches by calling the subroutine: **)
541       processPolyDataFn = OptionValue[ProcessPolyDataFn];
542       matchData = processPolyDataFn[cfFactorData, indexOffsets, cfg];
543
544       Return[matchData];
545
546   ]
547
548   ProcessSingleFactorPoly[poly_, polyIndex_, polyVar_, indexOffsets_,
549                           factorFn_, cfgOptions : OptionsPattern[]] :=
550   ProcessPolynomialWrapper[poly, polyIndex, polyVar, indexOffsets, factorFn,
551                            ProcessPolyDataFn -> ProcessSingleFactorPolyData,
552                            cfgOptions];
553
554   ProcessMultipleFactorPoly[poly_, polyIndex_, polyVar_, indexOffsets_,
555                             factorFn_, cfgOptions : OptionsPattern[]] :=
556   ProcessPolynomialWrapper[poly, polyIndex, polyVar, indexOffsets, factorFn,
557                            ProcessPolyDataFn -> ProcessMultipleFactorPolyData,
558                            cfgOptions];
559
560   getMultipleIndexOffsetFunction[uiList_, liList_, indexOffsetsList_,
561                                  valData_] :=
562   Module[{uiOffsets, liOffsets, ulIndexFn, getOffsetFns, uiFns, liFns,
563           ulOffsetFn, mapListFn1, mapListFn2, mapListFn12,
564           intermedFn, ulOffsetFn2},
565
566       (** : Error checking on the lengths of the input lists? : **)
567
568       PrintStatus["indexOffsetsList: ", indexOffsetsList];
569       uiOffsets = Map[(#1)[[1]]&, indexOffsetsList];
570       PrintStatus["uiOffsets: ", uiOffsets];
571
572       liOffsets = Map[(#1)[[2]]&, indexOffsetsList];
573
574       ulIndexFn[ulIndex_, ulOffset_] := (ulIndex + ulOffset * (#1))&;
575       getOffsetFns[indexList_, offsetList_] :=
576            MapIndexed[ulIndexFn[indexList[[ First[#2] ]], #1]&, offsetList];
577
578       uiFns = getOffsetFns[uiList, uiOffsets];
579       PrintStatus["I: ", uiFns];
580
581       liFns = getOffsetFns[liList, liOffsets];
582       PrintStatus["II: ", liFns];
583
584       ulOffsetFn = PackageMultipleTSeqData[uiFns, liFns, valData];
585       PrintStatus["III: ", ulOffsetFn];
586
587       (*x = Unique[xi];
588       mapListFn1 = Map[((#1)[x])&, #1]&;
589       mapListFn2 = Map[MapListFn1[#1]&, #1]&;
590       intermedFn = HoldForm[Evaluate[ReplacePart[ulOffsetFn,
591                     1 -> (mapListFn2[ ulOffsetFn[[1]] ])] /. (x -> #)]];
592       ulOffsetFn2 = Function @@ intermedFn; *)
593
594       (** : What these next few lines are actually used for in the code: **)
595       (** : Should transform a list that looks like                      **)
596       (** : {{{0 + 0 #1 &, 0 + 1 #1 &}}, _} or like                      **)
597       (** : {{{a + b #1 &, c + d #1 &}}, _} ...                          **)
598       (** : into the correct function syntax of a list that looks like   **)
599       (** : {{{0 + 0 #1, 0 + 1 #1}}, _}& or like                        **)
600       (** : {{{a + b #1, c + d #1}}, _}&                                **)
601
602       x = Unique[xi];
```

```
603       mapListFn12 = Map[(Map[((#1)[x])&, #1]&)[#1]&, #1]&;
604       ulOffsetFn2 = With[{uloFn = ulOffsetFn, mlFn12 = mapListFn12, xvar = x},
605                     Function @@
606                     HoldForm[Evaluate[ReplacePart[uloFn,
607                     1 -> (mapListFn12[ ulOffsetFn[[1]] ])] /. (xvar -> #)]] ];
608
609       PrintStatus["IV: ", ulOffsetFn2];
610
611       Return[ulOffsetFn2];
612
613  ]
614
615  Options[ProcessSingleFactorPolyData] = DefaultPkgConfigOptions;
616  ProcessSingleFactorPolyData[polyFactorData_, indexOffsets_,
617                              cfg : OptionsPattern[]] :=
618  Module[{cfListFactors, firstcfFactors, matches, cfi, curFactorData,
619          uIndexList, lIndexList, ulOffsetFn, testMemberFormFn, testOffsets,
620          remTermPos, remSeqTerms, matchData},
621
622       (*ConfigDebuggingMessages[{True, False, True}]; *)
623       PrintStatus["In Second Process Fn ... "];
624
625       (** : Later can pick the index with the fewest factor data terms ... : **)
626       cfListFactors = polyFactorData;
627       firstcfFactors = cfListFactors[[1]];
628       cfListFactors = Drop[cfListFactors, 1];
629
630       PrintDebug["firstcfFactors: ", firstcfFactors];
631       PrintDebug["cfListFactors: ", cfListFactors];
632
633       matches = {};
634       For[cfi = 1, cfi <= Length[firstcfFactors], cfi++,
635
636            curFactorData = firstcfFactors[[cfi]];
637            PrintVariable[curFactorData];
638
639            uIndexList = ExtractMultipleTSeqUpperIndices[curFactorData];
640            PrintVariable[uIndexList];
641
642            lIndexList = ExtractMultipleTSeqLowerIndices[curFactorData];
643            PrintVariable[lIndexList];
644            ulOffsetFn = getMultipleIndexOffsetFunction[uIndexList, lIndexList,
645                                                        indexOffsets, _];
646            PrintStatus["after I ", ulOffsetFn];
647
648            testMemberFormFn = MemberQ[#1, ulOffsetFn[ First[#2]] ]&;
649            testOffsets = MapIndexed[testMemberFormFn[#1, #2]&, cfListFactors];
650            PrintStatus["after II"];
651
652            If[!MemberQ[testOffsets, False],
653
654                 (** : Potential match, get the leftover sequence terms: **)
655                 remTermPos = MapIndexed[Position[#1, ulOffsetFn[First[#2]]]&,
656                                         cfListFactors];
657                 PrintMatchInfo["remTermPos (pre flatten): ", remTermPos];
658                 remTermPos = Flatten[remTermPos, 1];
659                 PrintMatchInfo["remTermPos (post flatten): ", remTermPos];
660
661                 remSeqTerms = MapIndexed[ExtractTSeqValue[
662                               cfListFactors[[ First[#2] ]][[#1]][[1]] ]&,
663                               remTermPos];
664                 remSeqTerms = Prepend[remSeqTerms,
665                                       ExtractTSeqValue[curFactorData]];
666
667                 (*matchData = {{uIndex, lIndex}, remSeqTerms};*)
668                 matchingULIndexData = ExtractTSeqULIndexData[
669                      PackageMultipleTSeqData[uIndexList, lIndexList, Null]
```

```
670                 ];
671                 matchData = PackagePolyIndexMatchData[matchingULIndexData,
672                                                       remSeqTerms];
673
674                 (*matchData = {{uIndex, lIndex}, remSeqTerms, Reverse[remSeqTerms]};*)
675                 PrintMatchInfo["matchData : ", matchData];
676                 matches = Append[matches, matchData];
677
678             ];
679
680         ];
681
682         PrintMatchInfo["full matches list: ", matches];
683         Return[matches]
684
685 ]
686
687 (**** : Begin FilterByRemSequenceMatches[...] helper functions : ****)
688
689 getPermsMapFn[seqEntry_] :=
690 Module[{seqData, uiIndexPerms},
691
692         seqData = seqEntry[[1]];
693         uiIndexPerms = Tuples[seqEntry[[2]]];
694         Return[Map[{#1, seqData}&, uiIndexPerms]];
695
696 ]
697
698 (**** : End FilterByRemSequenceMatches[...] helper functions : ****)
699
700 Options[FilterByRemSequenceMatches] = DefaultPkgConfigOptions;
701 FilterByRemSequenceMatches[matchData_, preProcFn_,
702                            cfgOptions : OptionsPattern[]] :=
703 Module[{localFilterBySeqDataFn, lastIndex, lastMatchData, nullSequenceValue,
704         mdi, prevMatchData, seqLenDiff, seqEqualsFn,
705         diffPrevIndexRemSeqTermsFn, fullPermsData},
706
707         (** : Returns a list of processed pairs of the form **)
708         (** : {SeqData, {UI Index Match Pairs}}:             **)
709         localFilterBySeqDataFn[mdIndex_, inputMatchData_] :=
710         Module[{extractSeqDataFn, extractUIDataFn, seqSplitFn, splitResults,
711                 procDataList, sindex, splitEntry, seqData, uiDataList},
712
713             PrintStatus["inputMatchData[[", mdIndex, "]]: ",
714                         inputMatchData[[mdIndex]]];
715
716             extractSeqDataFn = OptionValue[ExtractSequenceDataFunction];
717             extractUIDataFn = OptionValue[ExtractUIIndexDataFunction];
718             seqSplitFn = preProcFn[extractSeqDataFn[#1]]&;
719             splitResults = SplitBy[inputMatchData[[mdIndex]], seqSplitFn];
720             PrintDebug["localFilterBySeqDataFn[", mdIndex, "]: ", splitResults];
721
722             procDataList = {};
723             For[sindex = 1, sindex <= Length[splitResults], sindex++,
724                 splitEntry = splitResults[[sindex]];
725                 seqData = seqSplitFn[splitEntry[[1]]];
726                 uiDataList = Map[extractUIDataFn[#1]&, splitEntry];
727                 PrintDebug["=> sindex = ", sindex, ": ", uiDataList];
728                 procDataList = Append[procDataList, {seqData, uiDataList}];
729             ];
730
731             PrintDebug["localFilterBySeqDataFn[", mdIndex, "]: ", procDataList];
732             Return[procDataList];
733
734         ]; (* localFilterBySeqDataFn *)
735
736         lastIndex = Length[matchData];
```

```
737       lastMatchData = localFilterBySeqDataFn[lastIndex, matchData];
738
739       (* Store a list of lists for the last arguments: *)
740       lastMatchData = Map[{(#1)[[1]], {(#1)[[2]]}}&, lastMatchData];
741       PrintStatus["At mdi index = lastIndex (", lastIndex, "): ", lastMatchData];
742
743       nullSequenceValue = OptionValue[ReturnNullSequenceType];
744
745       For[mdi = lastIndex - 1, 1 <= mdi, mdi--,
746
747            (** : Return the empty set if there are no current : **)
748            (** : consistent sequence matches:                   **)
749            If[IsEmptySet[lastMatchData],
750                 Return[{}];
751            ];
752
753            prevMatchData = localFilterBySeqDataFn[mdi, matchData];
754            PrintStatus["At mdi index = ", mdi, " (I): ", prevMatchData];
755
756            seqLenDiff = lastIndex - mdi;
757            seqEqualsFn = (Drop[#1, -seqLenDiff] == #2)&;
758            diffPrevIndexRemSeqTermsFn[lastmd_] :=
759            Module[{lastmdSeq, prevSeqMatch},
760
761                 PrintDebug["==> lastmd: ", lastmd];
762                 lastmdSeq = lastmd[[1]];
763                 prevSeqMatch = Select[prevMatchData,
764                                       seqEqualsFn[lastmdSeq, (#)[[1]] ]&];
765                 If[Length[prevSeqMatch] == 0,
766                      Return[nullSequenceValue];
767                 ];
768
769                 prevSeqMatch = prevSeqMatch[[1]]; (* Select returns a list *)
770                 PrintDebug["==> prevSeqMatch: ", prevSeqMatch];
771                 Return[{lastmdSeq, Prepend[lastmd[[2]], prevSeqMatch[[2]] ]}];
772
773            ];
774            lastMatchData = Map[diffPrevIndexRemSeqTermsFn[#1]&, lastMatchData];
775            lastMatchData = Cases[lastMatchData, Except[nullSequenceValue]];
776            PrintStatus["At mdi index = ", mdi, " (II): ", lastMatchData];
777
778       ]; (* for mdi *)
779
780       PrintStatus[" ==== After mdi loop ==== "];
781       PrintDebug["lastMatchData: ", lastMatchData];
782
783       (** : Generate all possible permutations of the form :  **)
784       (** : {{UI Index Data Pairs}, Processed Sequence Data}: **)
785       PrintDebug["fullPermsData (pre): ", Map[getPermsMapFn[#1]&, lastMatchData]];
786       fullPermsData = Flatten[Map[getPermsMapFn[#1]&, lastMatchData], 1];
787       PrintDebug["fullPermsData (post): ", fullPermsData];
788
789       Return[fullPermsData];
790
791  ]
792
793  (**** : Begin VerifyFormulaMatches[...] helper functions : ****)
794
795  Options[computeIndexFormulaFn] = DefaultPkgConfigOptions;
796  computeIndexFormulaFn[seqData_, indexOffset_,
797                        cfgOptions : OptionsPattern[]] :=
798  Module[{seqStartIndex, formulaFnData, nullFn, formulaFnj, offsetFn,
799          fullIndexFn},
800
801       seqStartIndex = OptionValue[StartIndex];
802
803       formulaFnData = ComputeSequenceFormula[seqData, seqStartIndex,
```

```
804                                                   cfgOptions];
805       If[!formulaFnData[[1]],
806            nullFn = formulaFnData[[2]];
807            Return[{False, nullFn, nullFn, nullFn}];
808       ];
809
810       formulaFnj = formulaFnData[[2]];
811       offsetFn = (indexOffset * (#))&;
812       fullIndexFn = (offsetFn[#2] + formulaFnj[#1])&;
813
814       (** : Later, change these to transformation rules, or give an : **)
815       (** : extract data function for the indices ... :              **)
816       formulaFnData = {True, formulaFnj, offsetFn, fullIndexFn};
817
818       Return[formulaFnData];
819
820  ]
821
822  Options[computeRemSeqFormulaFn] = DefaultPkgConfigOptions;
823  computeRemSeqFormulaFn[rseqData_,
824                         cfgOptions : OptionsPattern[]] :=
825  Module[{requireRemSeqFormula, processRemSeqFormulas, reverseRemSeqIndices,
826          rsFormulaData, rsInitFormula, rsFormula},
827
828       requireRemSeqFormula = OptionValue[RequireMatchingRemSequenceFormula];
829       processRemSeqFormulas = OptionValue[ProcessRemSequenceFormulas];
830       reverseRemSeqIndices = OptionValue[ReverseRemSeqFormulaIndex];
831
832       If[requireRemSeqFormula || processRemSeqFormulas,
833
834            rsFormulaData = ComputeSequenceFormula[rseqData, 0, cfgOptions];
835            rsInitFormula = rsFormulaData[[2]];
836            rsFormula = rsInitFormula;
837            If[!reverseRemSeqIndices,
838                 rsFormula = (rsInitFormula[#2])&,      (* index i *)
839                 rsFormula = (rsInitFormula[#1 - #2])& (* index j-i *)
840            ];
841
842            Return[{rsFormulaData[[1]], rsFormula, rseqData}];
843
844       ];
845
846       Return[{True, GetFixedSequenceDataFn[rseqData, 0, #1]&, rseqData}];
847
848  ]
849
850  getUIIndexSeqFn[cfData_, indexPos_, factorPos_] :=
851  Module[{mapExtractFn},
852
853       mapExtractFn = (#1)[[factorPos]][[indexPos]]&;
854       Return[Map[mapExtractFn[#1]&, cfData]];
855
856  ]
857
858  (**** : End VerifyFormulaMatches[...] helper functions : ****)
859
860  Options[VerifyFormulaMatches] = DefaultPkgConfigOptions;
861  VerifyFormulaMatches[inputMatchData_, indexOffsets_,
862                       cfgOptions : OptionsPattern[]] :=
863  Module[{requireULIndexFormula, requireRemSeqFormula, noProcessSymmetricRemSeq,
864          reverseRemSeqIndices, seqFactorIndex, uiIndexOffset, liIndexOffset,
865          formulaMatches, mresult, matchResultData, coeffData,
866          upperIndexSeq, uiFn, lowerIndexSeq, liFn, remSeq, rsFn,
867          formulaData, numSeqFactors, uiFnsList, liFnsList,
868          breakOnFormulaError},
869
870       PrintDebug["inputMatchData: ", inputMatchData];
```

```
871
872         (** : The primary settings for verifying the index and : **)
873         (** : sequence formulas are matched:                    **)
874         requireULIndexFormula = OptionValue[RequireMatchingULIndexFormula];
875         requireRemSeqFormula = OptionValue[RequireMatchingRemSequenceFormula];
876         noProcessSymmetricRemSeq = OptionValue[AbortOnSymmetricRemSequence];
877
878         (** : Configure whether the remaining sequence index input is : **)
879         (** : i or j-i :                                               **)
880         reverseRemSeqIndices = OptionValue[ReverseRemSeqFormulaIndex];
881         SetOptions[computeRemSeqFormulaFn,
882                    ReverseRemSeqFormulaIndex -> reverseRemSeqIndices];
883
884         (** : Setup other configuration setting options: **)
885         (*PackageFormulaFn = OptionValue[PackageIntermediateFormulaDataFunction]; *)
886
887         (** : Setup sequence offset and position information: **)
888         numSeqFactors = Length[OptionValue[SequenceFactors]];
889
890         (** : Find which of the possible matches give consistent formulas: **)
891         formulaMatches = {};
892         For[mresult = 1, mresult <= Length[inputMatchData], mresult++,
893
894               matchResultData = inputMatchData[[mresult]];
895               PrintStatus["matchResultData (mresult = ", mresult, "): ",
896                           matchResultData];
897
898               (** : Process the remaining sequence formula for this match: **)
899               remSeq = matchResultData[[2]];
900               If[noProcessSymmetricRemSeq && SameQ[remSeq, Reverse[remSeq]],
901                     Continue[];
902               ];
903
904               rsFn = computeRemSeqFormulaFn[remSeq, cfgOptions];
905               If[requireRemSeqFormula && !rsFn[[1]],
906                     Continue[];
907               ];
908
909               PrintMatchInfo["Rem Seq -> ",
910                              rsFn[[2]] @@ Map[SymbolName[#1]&, {j,i}]];
911
912               (** : Process the upper and lower index formulas for each : **)
913               (** : of the sequence factors                              : **)
914               (*coeffData = matchResultData[[1]];*)
915               coeffData = ExtractTSeqULIndexData[matchResultData];
916               uiFnsList = {};
917               liFnsList = {};
918               breakOnFormulaError = False;
919               For[seqFactorIndex = 1, seqFactorIndex <= numSeqFactors,
920                   seqFactorIndex++,
921
922                    uiIndexOffset = indexOffsets[[seqFactorIndex]][[1]];
923                    upperIndexSeq = getUIIndexSeqFn[coeffData, 1, seqFactorIndex];
924                    uiFn = computeIndexFormulaFn[upperIndexSeq, uiIndexOffset,
925                                                 cfgOptions];
926
927                    If[requireULIndexFormula && !uiFn[[1]],
928                          breakOnFormulaError = True;
929                          Break[];
930                    ];
931
932                    liIndexOffset = indexOffsets[[seqFactorIndex]][[2]];
933                    lowerIndexSeq = getUIIndexSeqFn[coeffData, 2, seqFactorIndex];
934                    liFn = computeIndexFormulaFn[lowerIndexSeq, liIndexOffset,
935                                                 cfgOptions];
936
937                    If[requireULIndexFormula && !liFn[[1]],
```

```
938                     breakOnFormulaError = True;
939                     Break[];
940                 ];
941
942                 uiFnsList = Append[uiFnsList, uiFn[[4]] ];
943                 liFnsList = Append[liFnsList, liFn[[4]] ];
944
945                 PrintDebug["coeffData: ", coeffData];
946                 PrintDebug["upper seq (index=", seqFactorIndex, "): ",
947                         upperIndexSeq];
948                 PrintDebug["lower seq (index=", seqFactorIndex, "): ",
949                         lowerIndexSeq];
950                 PrintDebug["remSeq: ", remSeq];
951
952                 PrintMatchInfo["Upper Index (", seqFactorIndex, ") -> ",
953                         uiFn[[4]] @@ Map[SymbolName[#1]&, {j,i}]];
954                 PrintMatchInfo["Lower Index (", seqFactorIndex, ") -> ",
955                         liFn[[4]] @@ Map[SymbolName[#1]&, {j,i}]];
956
957         ]; (* For seqFactorIndex *)
958
959         If[breakOnFormulaError,
960             Continue[];
961         ];
962
963         (*formulaData = {{{uiFn[[4]], liFn[[4]]}}, rsFn[[2]], rsFn[[3]]};*)
964         ulIndexFnsData = PackageMultipleTSeqData[uiFnsList, liFnsList, Null];
965         ulIndexFnsData = ExtractTSeqULIndexData[ulIndexFnsData];
966         formulaData = {ulIndexFnsData, rsFn[[2]], rsFn[[3]]};
967         formulaMatches = Append[formulaMatches, formulaData];
968
969     ]; (* For mresult *)
970
971     PrintDebug["formulaMatches: ", formulaMatches];
972     Return[formulaMatches];
973
974 ]
975
976 (**** : Begin GuessPolynomialSequence[...] helper functions : ****)
977
978 getSequenceFactorFnGenerators[seqFactorsList_] :=
979 Module[{seqFactorFns, findex, seqFactorID,
980         seqFactorFn, seqMetaData, seqGenFn},
981
982     seqFactorFns = {};
983     For[findex = 1, findex <= Length[seqFactorsList], findex++,
984
985         seqFactorID = seqFactorsList[[findex]];
986         seqMetaData = QuerySequenceMetaData[seqFactorID];
987         seqGenFn = (GeneratorFunction) /. seqMetaData;
988         PrintDebug["seqGenFn: ", seqGenFn];
989         seqFactorFns = Append[seqFactorFns, seqGenFn];
990
991     ]; (* For findex *)
992
993     Return[seqFactorFns];
994
995 ]
996
997 getSequenceFactorFormulaTerm[matchData_, seqFactorFnList_] :=
998 Module[{seqFactorCfList, sfindex, seqFactorFn, uiInput, liInput,
999         seqFactorCf, seqFactorCfFn},
1000
1001     seqFactorCfList = {};
1002     For[sfindex = 1, sfindex <= Length[seqFactorFnList], sfindex++,
1003
1004         seqFactorFn = seqFactorFnList[[sfindex]];
```

```
1005            uiInput = matchData[[1]][[sfindex]][[1]];
1006            liInput = matchData[[1]][[sfindex]][[2]];
1007            seqFactorCf = seqFactorFn[uiInput[#1, #2], liInput[#1, #2]];
1008            seqFactorCfList = Append[seqFactorCfList, seqFactorCf];
1009
1010        ];
1011
1012        seqFactorCfFn = Times @@ seqFactorCfList;
1013        Return[seqFactorCfFn];
1014
1015    ]
1016
1017    (**** : End GuessPolynomialSequence[...] helper functions : ****)
1018
1019    Options[GuessPolynomialSequence] = DefaultPkgConfigOptions;
1020    GuessPolynomialSequence[polyseq_, polyVar_,
1021                            cfgOptions : OptionsPattern[]] :=
1022    Module[{userGuessFn, seqStartIndex, seqFactorsList, factorFn,
1023            procPolySeq, xvarPowMins, polyMaxDegrees, seqLength,
1024            pindex, polyIndex, initPolyForm, procPolyData, newPoly,
1025            minPolyDegree, maxPolyDegree, polyVarFactorPow,
1026            testIndexOffsetParams, indexMultiples, offsetValues, offsetPairs,
1027            matches, oindex, indexOffsets, processFnArgs, processFn,
1028            mapProcessFn, fullMatchData, remSeqPreprocessFns, ppi,
1029            ppFn, md, md2Matches, polyDegreeSeq, degreeFormula,
1030            seqFactorFns, matchingFormulaFns, mindex,
1031            seqFactorFn, uiInput, liInput, remSeqFormula,
1032            seqDegreeFormula, pvar, powOffset, displayVars, seqFactorCf,
1033            remSeqFn, polyPowTerm, userGuessTerm, innerSumTerms, sumBounds,
1034            sumFormula, printFormulasQ, headerTextStr, printHeaderStr,
1035            polyFormulaTerms, pLHS, pEQ, pRHS, polyFormulaStyle, printBullet,
1036            remSeqDataList, computeSeqDiffs, seqDiffs, seqDiffsCorrect,
1037            numPolySeqElements, numSeqFactorRows, clearLocalSeqData,
1038            formulaCountMax, displayVarNames},
1039
1040        (** : Pre-processing options and check other runtime options: **)
1041        If[OptionValue[EnableDebugging],
1042            EnableDebugging[],
1043            DisableDebugging[];
1044        ];
1045
1046        GuessPolySequenceFormulas`PkgConfig`AllowSymbolicData =
1047            OptionValue[AllowSymbolicData];
1048
1049        (** : Get configuration option settings: **)
1050        userGuessFn = OptionValue[UserGuessFunction];
1051        seqStartIndex = OptionValue[StartIndex];
1052        seqFactorsList = OptionValue[SequenceFactors];
1053        factorFn = OptionValue[FactorFunction];
1054
1055        (** : Setup the local sequence handling if the user did not : **)
1056        (** : specifiy an alternate handling function:                **)
1057        If[factorFn == Null,
1058            If[!BuildLocalSequenceData[cfgOptions],
1059                Return[{}];
1060            ];
1061            (*factorFn = FactorIntegerBySequence; *)
1062            factorFn = FactorIntegerBySequences;
1063        ];
1064
1065        PrintStatus["In this function ... ", "StartIndex->", seqStartIndex];
1066
1067        (** : Pre-process the input polynomial sequence: **)
1068        procPolySeq = {};
1069        xvarPowMins = {};
1070        polyMaxDegrees = {};
1071        seqLength = Length[polyseq];
```

```
1072        For[pindex = 1, pindex <= Length[polyseq], pindex++,
1073
1074            polyIndex = seqStartIndex + pindex - 1;
1075            initPolyForm = polyseq[[pindex]];
1076            procPolyData = PreProcessPolynomialData[initPolyForm, polyIndex,
1077                                                    polyVar, userGuessFn,
1078                                                    cfgOptions];
1079            If[Length[procPolyData] == 0,
1080                Return[{}];
1081            ];
1082
1083            newPoly = (ModPolyFn /. procPolyData);
1084            procPolySeq = Append[procPolySeq, newPoly];
1085            minPolyDegree = Exponent[newPoly, polyVar, Min];
1086            maxPolyDegree = Exponent[newPoly, polyVar];
1087            xvarPowMins = Append[xvarPowMins, minPolyDegree];
1088            polyMaxDegrees = Append[polyMaxDegrees, maxPolyDegree];
1089
1090        ];
1091
1092        PrintDebug["procPolySeq: ", procPolySeq];
1093        PrintDebug["min exponents list: ", xvarPowMins];
1094
1095        (** : Figure out if can factor out a multiple of the variable x^pow: **)
1096        polyVarFactorPow = Min[xvarPowMins];
1097        If[polyVarFactorPow > 0,
1098            procPolySeq = Cancel[procPolySeq / Power[polyVar, polyVarFactorPow]];
1099            polyMaxDegrees = polyMaxDegrees - polyVarFactorPow;
1100            PrintStatus[StringForm["Canceled factor of ``^", polyVar],
1101                        polyVarFactorPow, ": " procPolySeq];
1102            PrintStatus["Call Message for this status ... "];
1103        ];
1104
1105        (** : Get the sets of index offset parameters: **)
1106        (*testIndexOffsetParams = { {{0, 1}} };*)              (* initial setting *)
1107        (*testIndexOffsetParams = { {{0, 1}}, {{0, -1}} };*) (* initial setting *)
1108        (*testIndexOffsetParams = { {{0, -1}, {0, 1}} };*)    (* initial setting *)
1109        testIndexOffsetParams = OptionValue[IndexOffsetPairs];
1110        If[testIndexOffsetParams == Null,
1111            indexMultiples = OptionValue[IndexMultiples];
1112            offsetValues = Union[Flatten[Map[{#1, -#1}&, indexMultiples]]];
1113            offsetPairs = Tuples[offsetValues, 2];
1114            testIndexOffsetParams = Tuples[offsetPairs, Length[seqFactorsList]];
1115        ]; (* otherwise, use the user-defined setting *)
1116
1117        (** : Loop over the possible offset and input options : **)
1118        matches = {};
1119        For[oindex = 1, oindex <= Length[testIndexOffsetParams], oindex++,
1120
1121            indexOffsets = testIndexOffsetParams[[oindex]];
1122            PrintStatus["indexOffsets: ", indexOffsets];
1123
1124            processFnArgs = {#1, seqStartIndex + First[#2] - 1,
1125                             polyVar, indexOffsets, factorFn,
1126                             cfgOptions}&;
1127
1128            processFn = ProcessSingleFactorPoly[##]&;
1129            mapProcessFn = (processFn @@ processFnArgs[#1, #2])&;
1130            fullMatchData = MapIndexed[mapProcessFn[#1, #2]&, procPolySeq];
1131            PrintMatchInfo["fullMatchData: ", fullMatchData];
1132
1133            If[IsEmptySet[fullMatchData],
1134                Continue[];
1135            ];
1136
1137            (** : Process this information: **)
1138            remSeqPreprocessFns = {Identity, Reverse};
```

```
1139            For[ppi = 1, ppi <= Length[remSeqPreprocessFns], ppi++,
1140
1141                ppFn = remSeqPreprocessFns[[ppi]];
1142                PrintStatus[StringForm["Pre matches: [oi=``, ppi=``]",
1143                            oindex, ppi], matches];
1144                md = FilterByRemSequenceMatches[fullMatchData, ppFn,
1145                                                cfgOptions];
1146                PrintStatus["Post matches: ", matches];
1147                PrintMatchInfo["After FilterByRemSequenceMatches (md): ", md];
1148
1149                If[ppFn === Identity,
1150                    SetOptions[VerifyFormulaMatches,
1151                               ReverseRemSeqFormulaIndex -> False];
1152                    SetOptions[VerifyFormulaMatches,
1153                               AbortOnSymmetricRemSequence -> False];
1154                ];
1155                If[ppFn === Reverse,
1156                    SetOptions[VerifyFormulaMatches,
1157                               ReverseRemSeqFormulaIndex -> True];
1158                    SetOptions[VerifyFormulaMatches,
1159                               AbortOnSymmetricRemSequence -> True];
1160                ];
1161
1162                md2Matches = VerifyFormulaMatches[md, indexOffsets,
1163                                                  cfgOptions];
1164                PrintStatus["matches: ", matches];
1165                PrintStatus["md2Matches: ", md2Matches];
1166                matches = Join[matches, md2Matches];
1167                PrintMatchInfo["After FilterByRemSequenceMatches: ",
1168                               Short[matches, 2]];
1169
1170            ];
1171
1172        ];
1173        PrintDebug["!!!!!!!!!!! [After main loop] ==========="];
1174
1175        (** : Get the sequence of polynomial degrees: **)
1176        polyDegreeSeq = Map[Exponent[#1, polyVar]&, procPolySeq];
1177        degreeFormula = ComputeSequenceFormula[polyDegreeSeq,
1178                                               seqStartIndex, cfgOptions];
1179
1180        If[!degreeFormula[[1]],
1181            Message[GPSFPkgMsgs::GuessMultipleFactorPolySequence::PolySeqDegree];
1182            If[Length[polyseq] < 6,
1183                Message[GPSFPkgMsgs::Warnings::InsufficientSeqElements,
1184                        Length[polyseq], 8];
1185            ];
1186            Return[{}];
1187        ];
1188        PrintStatus["sequence degreeFormula: ",
1189                    degreeFormula[[2]] @@ Map[SymbolName[#1]&, {j,i}]];
1190
1191        (** : Format and print the computed formulas:                    **)
1192        (** : Later, create separate formula creation routines (options) : **)
1193        (** : and printing functions                                      : **)
1194        Off[Sum::itraw]; (* temporarily disable this message for
1195                            formula creation and printing *)
1196
1197        (** : Get list of sequence factor function generators: **)
1198        (*seqFactorFns = Map[QuerySequenceMetaDataByProperty[#1,
1199                             GeneratorFunction]&, seqFactorsList]; *)
1200        seqFactorFns = getSequenceFactorFnGenerators[seqFactorsList];
1201
1202        matchingFormulasFns = {};
1203        formulaCountMax = Min[Length[matches], OptionValue[LimitFormulaCount]];
1204        For[mindex = 1, mindex <= formulaCountMax, mindex++,
1205
```

```
(** : Get the components to assemble the : **)
(** : matching formula functions:        : **)
seqFactorCf = getSequenceFactorFormulaTerm[matches[[mindex]],
                                           seqFactorFns];
remSeqFormula = matches[[mindex]][[2]];
seqDegreeFormula = degreeFormula[[2]];

(** : Pretty printing of the formula variables without context: **)
pvar = polyVar;
powOffset = polyVarFactorPow;

displayVarNames = {j, i};
If[Length[OptionValue[DisplayVars]] == 2,
    displayVarNames = OptionValue[DisplayVars];
];
displayVarNames = Append[displayVarNames, pvar];
(*displayVars = Map[SymbolName[#1]&, {j, i, pvar}] *)
displayVars = Map[SymbolName[#1]&, displayVarNames];

(** : Create th formula function: **)
(** : Input order: {j, i, x} (Poly index, Sum index, Poly Var) : **)
remSeqFn = remSeqFormula[seqDegreeFormula[#1], #2];
polyPowTerm = Power[#3, #2 + powOffset];
userGuessTerm = userGuessFn[#1, #2];
innerSumTerms = Times @@ {seqFactorCf, remSeqFn, userGuessTerm,
                          polyPowTerm};
sumBounds = {#2, 0, seqDegreeFormula[#1]};

sumFormula = With[{st = innerSumTerms, sb = sumBounds},
                  Function @@ HoldForm[Sum[st, sb]]];

matchingFormulasFns = Append[matchingFormulasFns, sumFormula];

(** : Print the formulas to the notebook (or skip option): **)
printFormulasQ = OptionValue[PrintFormulas];
If[!printFormulasQ,
    Continue[];
];

(** : Print a separate header for each match: **)
headerTextStr = StringForm["Found Matching Formula #`` / ``:",
                           mindex, Length[matches]];
printHeaderStr = "========== " <>
                 ToString[headerTextStr] <>
                 " ==========";
Print[Style[printHeaderStr, White, Bold,
            Background -> Darker[Blue, 0.36], FontSize -> 14]];

(** : Display the polynomial formula: **)
polyFormulaTerms = TraditionalForm[
    (Subscript["Poly", #1][#3]& @@ displayVars)
    \[RightTeeArrow] (sumFormula @@ displayVars)
];

pLHS = TraditionalForm[Subscript["Poly", #1][#3]& @@ displayVars];
pEQ = " \[RightTeeArrow] ";
pRHS = TraditionalForm[sumFormula @@ displayVars];
polyFormulaStyle = Style[#1, Bold, FontSize -> 15]&;
Print[" ", polyFormulaStyle[pLHS], polyFormulaStyle[pEQ],
          polyFormulaStyle[pRHS]];

(** : Output metadata as a bulleted list: **)
printBullet = Print[" ", "\[FilledRightTriangle]", " ",
                    Style[#1, Purple, Bold, FontSize -> 12], ##2]&;

If[OptionValue[PrintLaTeXFormulas],
    printBullet["Latex Formula Output: ", Null];
```

```
1273            ];
1274
1275            remSeqDataList = matches[[mindex]][[3]];
1276            printBullet["Remaining Sequence Data: ",
1277                        Shallow[remSeqDataList, {Infinity, 10}]];
1278
1279            printBullet["User Function: ",
1280                 TraditionalForm[Subscript["U", "guess"][#1, #2]&
1281                                 @@ displayVars], " \[LongEqual] ",
1282                 TraditionalForm[userGuessFn[#1, #2]& @@ displayVars]];
1283            printBullet["Formula Function: ",
1284                 TraditionalForm[Subscript["PolyFormula",
1285                                 StringForm["index=``", mindex]][##]&
1286                                 @@ displayVars]
1287            ];
1288            (*printBullet["Full Form", " \[LongRightArrow] ",
1289                          FullForm[sumFormula @@ displayVars]]; *)
1290
1291            computeSeqDiffs = OptionValue[DiffPolySequenceFormulas];
1292            If[!computeSeqDiffs,
1293                 Continue[];
1294            ];
1295
1296            seqDiffs = DiffSequenceFormula[polyseq, polyVar, sumFormula,
1297                                           cfgOptions];
1298            seqDiffsCorrect = (Count[seqDiffs, False] == 0);
1299            printBullet["Sequence Formula Diffs: ",
1300                        Short[seqDiffs, 0.5],
1301                        Which[seqDiffsCorrect,
1302                              Style[" [\[Checkmark]]", Medium, Bold, Green],
1303                              !seqDiffsCorrect,
1304                              Style[" [\[ScriptX]]", Medium, Bold, Red]
1305                        ]
1306            ];
1307
1308       ];
1309       On[Sum::itraw]; (* turn this message back on *)
1310
1311       (** : Issue possible warning messages if no formulas for the : **)
1312       (** : sequence are found:                                     **)
1313       If[IsEmptySet[matchingFormulasFns] || OptionValue[IssueAllWarnings],
1314
1315            numPolySeqElements = Length[polyseq];
1316            numSeqFactorRows = OptionValue[TriangularSequenceNumRows];
1317            If[numPolySeqElements < 6,
1318                 Message[GPSFPkgMsgs::Warnings::InsufficientSeqElements,
1319                         numPolySeqElements, 8];
1320            ];
1321            If[numSeqFactorRows < (numPolySeqElements + 4),
1322                 Message[GPSFPkgMsgs::Warnings::InsufficientFactorData,
1323                         numSeqFactorRows, numPolySeqElements + 4];
1324            ];
1325
1326       ]; (* end warning messages *)
1327
1328       (** : Clean up local sequence data storage: **)
1329       clearLocalSeqData = OptionValue[ClearLocalSequenceData];
1330       If[clearLocalSeqData,
1331            PkgSequenceData = {};
1332            PkgNumSequenceFactors = 0;
1333       ];
1334
1335       Return[matchingFormulasFns];
1336
1337  ]
1338
1339  End[] (* Private *)
```

```
1340
1341 (*************************************************************************)
1342 (**** : Perform pretty printing of the package details and          : ****)
1343 (**** : revsision information if the package is loaded in a notebook: ****)
1344 (*************************************************************************)
1345 packageInfoStr = "" <>
1346       GuessPolySequenceFormulas`PackageName <> ":\n" <>
1347       GuessPolySequenceFormulas`PackageShortDesc <> "\n" <>
1348       "Author: " <> GuessPolySequenceFormulas`PackageAuthor <> "\n" <>
1349       "Package Version: " <> GuessPolySequenceFormulas`PackageVersion;
1350
1351 If[$Notebooks,
1352       CellPrint[Cell[packageInfoStr, "Print",
1353                      FontColor -> RGBColor[0, 0, 0],
1354                      FontSize -> 12,
1355                      CellFrame -> 0.5,
1356                      CellFrameColor -> Purple,
1357                      Background -> LightBlue,
1358                      CellFrameMargins -> 14
1359                     ]
1360                ],
1361       Print[packageInfoStr];
1362 ];
1363
1364 EndPackage[]
1365
1366 (*************************************************************************)
1367 (*************************************************************************)
1368 (*************************************************************************)
1369 (*************************************************************************)
```

A.3 GuessSequenceData.m

Mathematica Source Code

```
1  (*************************************************************************)
2  (*************************************************************************)
3  (******************** : GuessSequenceData.m:          ********************)
4  (*************************************************************************)
5  (*************************************************************************)
6
7  BeginPackage["GuessSequenceData`"]
8
9  (*************************************************************************)
10 (**** : Package revision, metadata information, and settings:         ****)
11 (*************************************************************************)
12
13
14 LocalPackageName = "GuessSequenceData.m";
15 LocalPackageVersion = "2014.04.28-v5";
16 LocalPackageAuthor = "Maxie D. Schmidt";
17 LocalPackageShortDesc =
18       "Default package to provide several built-in sequences for the " <>
19       "GuessPolySequenceFormulas.m software package."
20
21 GSDPkg::FormattingData::NewlineString = "\n";
22 GSDPkg::FormattingData::ListBulletDelim = "  \[FilledRightTriangle] ";
23
24 (*************************************************************************)
25 (**** : Create default usage information for the package functions:   ****)
26 (**** : Provide detailed usage information for the package functions: ****)
27 (*************************************************************************)
28
29 (**** : Utilities / sequence lookup functions in the package: ****)
30 LookupSequenceKey::usage =
31       "LookupSequenceKey[seqID]: (Intended for internal use by the " <>
```

```
32        "GuessPolySequenceFormulas.m and local GuessSequenceData.m " <>
33        "packages)" <>
34        GSDPkg::FormattingData::NewlineString <>
35        "Used to obtain the internal index for the lookup table that " <>
36        "stores the metadata information about the sequences " <>
37        "(as specified by the short or long seqID key input to the " <>
38        "function) supported by the GuessSequenceData subpackage." <>
39        "The sequences currently supported by thie package are listed by "
40        "running the local package function ListSupportedSequences[].";
41
42  QuerySequenceMetaData::usage =
43        "QuerySequenceMetaData[seqID]:" <>
44        GSDPkg::FormattingData::NewlineString <>
45        "Returns the complete listing of metadata options stored " <>
46        "locally by the package for the sequence specified by the " <>
47        "(short or long) form of the seqID key " <>
48        "input to the function. A complete listing of the short and " <>
49        "long forms of the seqID keys currently supported by the " <>
50        "package can be viewed by running the local package function " <>
51        "ListSupportedSequences[]." <>
52        GSDPkg::FormattingData::NewlineString <>
53        "The sequence information returned is a listing of sequence "
54        "transformation rules corresponding to the public options " <>
55        "documented in the package source below." <>
56        "The listing of these options may also be viewed by " <>
57        "querying the usage from the Mathematica help lookup for the " <>
58        "function: ?QuerySequenceMetaDataByProperty" <>
59        GSDPkg::FormattingData::NewlineString <>
60        "Example Usage:" <>
61        GSDPkg::FormattingData::NewlineString <>
62        GSDPkg::FormattingData::ListBulletDelim <>
63        "QuerySequenceMetaData[\"S1\"]" <>
64        GSDPkg::FormattingData::NewlineString <>
65        GSDPkg::FormattingData::ListBulletDelim <>
66        "seqFn = (GeneratorFunction) /. QuerySequenceMetaData[\"S2\"];" <>
67        "Table[seqFn[n, k], {n,0,10}, {k,0,10}] // TableForm" <>
68        GSDPkg::FormattingData::NewlineString <>
69        GSDPkg::FormattingData::ListBulletDelim <>
70        "{Desc, NumInputs, OEISRefs} /. " <>
71        "QuerySequenceMetaData[\"EulerianE1\"]";
72
73  QuerySequenceMetaDataByProperty::usage =
74        "QuerySequenceMetaDataByProperty[seqID, mdataProp]:" <>
75        GSDPkg::FormattingData::NewlineString <>
76        "The function returns the metadata associated with the sequence " <>
77        "key seqID for the property mdataProp only. " <>
78        "Sequence Properties include: " <>
79        "LongSeqID, ShortSeqID, Desc, NumInputs, " <>
80        "GeneratorFunction, RowRangeFunction, " <>
81        "FormulaDisplay, LatexDisplay, Parameters, " <>
82        "Comments, OEISRefs, PrintRefs, WebRefs, OtherRefs, " <>
83        "RevisionInfo, Implementation, ImplNotes, OtherMetaData." <>
84        "The related package function QuerySequenceMetaData[seqID] " <>
85        "returned a complete listing of all metadata options for the " <>
86        "sequence in the form of list of replacement rules." <>
87        GSDPkg::FormattingData::NewlineString <>
88        "Example Usage:" <>
89        GSDPkg::FormattingData::NewlineString <>
90        GSDPkg::FormattingData::ListBulletDelim <>
91        "descStr = QuerySequenceMetaDataByProperty[\"StirlingS2\",Desc];" <>
92        "Print[\"Description of Sequence S2: \", descStr];" <>
93        GSDPkg::FormattingData::NewlineString <>
94        GSDPkg::FormattingData::ListBulletDelim <>
95        "seqFn = QuerySequenceMetaDataByProperty[\"S1\", GeneratorFunction];" <>
96        "Table[Power[-1,n-k]*seqFn[n, k] - StirlingS1[n,k], " <>
97        "{n,0,8}, {k,0,8}] // TableForm";
98
```

```
ListSequenceMetaData::usage =
    "ListSequenceMetaData[seqID, fullInfoQ:False, numPrintRows:6]:" <>
    GSDPkg::FormattingData::NewlineString <>
    "Provides a short summary of the sequence metadata information " <>
    "when fullInfoQ is False, or a complete listing of the local " <>
    "sequence information when the second parameter to the "
    "function is set to True. The third paremeter numPrintRows " <>
    "specifies how many rows of the triangular sequence values to " <>
    "print in the display returned by the function."
    GSDPkg::FormattingData::NewlineString <>
    "Example Usage:" <>
    GSDPkg::FormattingData::NewlineString <>
    GSDPkg::FormattingData::ListBulletDelim <>
    "ListSequenceMetaData[\"BinomialSquared\"];" <>
    GSDPkg::FormattingData::NewlineString <>
    GSDPkg::FormattingData::ListBulletDelim <>
    "ListSequenceMetaData[\"E2\",False,10];";

ListSupportedSequences::usage =
    "ListSupportedSequences[]:" <>
    GSDPkg::FormattingData::NewlineString <>
    "Displays a summary of all of the triangular sequences " <>
    "supported and/or implemented by the " <>
    "GuessSequenceData subpackage.";

(**** : Declare public options / replacement rules for the : ****)
(**** : built-in sequence metadata stored in the package:    ****)
LongSeqID::usage          = "GuessSequenceData subpackage metadata option.";
ShortSeqId::usage         = "GuessSequenceData subpackage metadata option.";
Desc::usage               = "GuessSequenceData subpackage metadata option.";
NumInputs::usage          = "GuessSequenceData subpackage metadata option.";
StartRowIndex::usage      = "GuessSequenceData subpackage metadata option.";
AllowZeros::usage         = "GuessSequenceData subpackage metadata option.";
GeneratorFunction::usage  = "GuessSequenceData subpackage metadata option.";
RowRangeFunction::usage   = "GuessSequenceData subpackage metadata option.";
FormulaDisplay::usage     = "GuessSequenceData subpackage metadata option.";
LatexDisplay::usage       = "GuessSequenceData subpackage metadata option.";
Parameters::usage         = "GuessSequenceData subpackage metadata option.";
Comments::usage           = "GuessSequenceData subpackage metadata option.";
OEISRefs::usage           = "GuessSequenceData subpackage metadata option.";
PrintRefs::usage          = "GuessSequenceData subpackage metadata option.";
WebRefs::usage            = "GuessSequenceData subpackage metadata option.";
OtherRefs::usage          = "GuessSequenceData subpackage metadata option.";
RevisionInfo::usage       = "GuessSequenceData subpackage metadata option.";
Implementation::usage     = "GuessSequenceData subpackage metadata option.";
ImplNotes::usage          = "GuessSequenceData subpackage metadata option.";
OtherMetaData::usage      = "GuessSequenceData subpackage metadata option.";

(*************************************************************************)
(**** : Package error handling and messages:                         ****)
(*************************************************************************)
GuessSequenceData::ErrorMsgs::InvalidSequenceID =
    "Invalid sequence ID `1`";
GuessSequenceData::ErrorMsgs::InvalidMetadataOption =
    "Invalid metadata property `1`";

GuessSequenceData::WarningMsgs::MultipleSeqKeyMatches =
    "Multiple sequence match ID `1` ... using first match key `2`";

Begin["`Private`"]

(**** : Built-in standard functions supported by the (sub)package:      ****)
(**** : Index Format: {LongIDKey, ShortIDKey, Lookup table ID, ValidQ}: ****)
PkgSupportedSequenceIDKeys = {
    {"StirlingS1",          "S1",          "SeqIDKey:S1:",          True},
    {"Stirlings1",          "s1",          "SeqIDKey:s1:",          True},
    {"StirlingS2",          "S2",          "SeqIDKey:S2:",          True},
```

```
166       {"EulerianE1",          "E1",          "SeqIDKey:E1:",          True},
167       {"EulerianE2",          "E2",          "SeqIDKey:E2:",          True},
168       {"Binomial",            "Binom",       "SeqIDKey:Binom:",       True},
169       {"BinomialSquared",     "Binom2",      "SeqIDKey:BinomPow2:",   True},
170       {"BinomialSymmetric",   "BinomSym",    "SeqIDKey:BinomSym:",    True},
171       {"Documentation",       "ExampleFn",   "SeqIDKey:Example:",     False}
172  };
173
174  LookupSequenceKey[seqID_] :=
175  Module[{posValues, seqInfoIndices},
176
177       posValues = Position[PkgSupportedSequenceIDKeys, seqID];
178       seqInfoIndices = Map[First[#1]&, posValues];
179       Return[ Map[PkgSupportedSequenceIDKeys[[#1]][[3]]&, seqInfoIndices] ];
180
181  ]
182
183  QuerySequenceMetaData[seqID_] :=
184  Module[{}, (*{seqLookupKeys, seqLookupKey, mdataIndexPos, mdataIndex}, *)
185
186       seqLookupKeys = LookupSequenceKey[seqID];
187       If[Length[seqLookupKeys] == 0,
188            Message[GuessSequenceData::ErrorMsgs::InvalidSequenceID, seqID];
189            Return[{}];
190       ];
191
192       If[Length[seqLookupKeys] > 1,
193            Message[GuessSequenceData::WarningMsgs::MultipleSeqKeyMatches,
194                 seqID, seqLookupKeys[[1]]];
195       ];
196       seqLookupKey = seqLookupKeys[[1]];
197
198       mdataIndexPos = Position[FullSequencesMetaData, seqLookupKey, 2, 1];
199       If[Length[mdataIndexPos] == 0,
200            Return[{}];
201       ];
202
203       (** : Format: { {index, 1} }: **)
204       mdataIndex = mdataIndexPos[[1]][[1]];
205       Return[Drop[FullSequencesMetaData[[mdataIndex]], 1]];
206
207  ]
208
209  QuerySequenceMetaDataByProperty[seqID_, mdataProp_] :=
210  Module[{}, (*{seqMetaData, rProp}, *)
211
212       seqMetaData = QuerySequenceMetaData[seqLookupKey];
213       If[Length[seqMetaData] == 0,
214            Return[Null];
215       ];
216
217       rProp = (mdataProp) /. seqMetaData;
218       If[rProp == mdataProp,
219            Message[GuessSequenceData::ErrorMsgs::InvalidMetadataOption,
220                 mdataProp];
221            Return[Null];
222       ];
223       Return[rProp];
224
225  ]
226
227  ListSequenceMetaData[seqID_, fullInfoQ_:False,
228                       numPrintRows_:6] :=
229  Module[{},
230
231       seqProps = {LongSeqID, ShortSeqID, Desc, NumInputs};
232       If[fullInfoQ,
```

```
233             remProps = {FormulaDisplay, LatexDisplay, Parameters,
234                         Comments, OEISRefs, PrintRefs, WebRefs, OtherRefs,
235                         RevisionInfo, Implementation, ImplNotes,
236                         OtherMetaData};
237             seqProps = Append[seqProps, remProps];
238         ];
239
240         seqGenFn = QuerySequenceMetaDataByProperty[seqID, GeneratorFunction];
241         rowRangeFn = QuerySequenceMetaDataByProperty[seqID, RowRangeFunction];
242
243         Print["ListSequenceMetaData: Implementation later "];
244         Print["Get list of sequence values ... "];
245 ]
246
247 ListSupportedSequences[overrideIsValidTag_:False] :=
248 Module[{},
249
250         pkgSeqKeyData = PkgSupportedSequenceIDKeys;
251         Print["ListSupportedSequences[]: Implement later ... "];
252         Print["Return a list of {LongKey, ShortKey} pairs ... "];
253
254 ]
255
256 (** : Previous, older versions of the Striling number triangles: **)
257 SequenceGeneratorS1[n_, k_] := Power[-1, n-k] * StirlingS1[n, k];
258 SequenceGeneratorS1[n_, k_] := Abs[StirlingS1[n, k]];
259 SequenceGeneratorS2[n_, k_] := StirlingS2[n, k];
260
261 SequenceGeneratorS1[n_, k_] := SeqFnS1[n, k];
262 SequenceGeneratorS1[n_, k_] := StirlingS1[n, k]
263 SequenceGeneratorS2[n_, k_] := SeqFnS1[n, k];
264 SequenceGeneratorE1[n_, k_] := SeqFnE1[n, k];
265 SequenceGeneratorE2[n_, k_] := SeqFnE2[n, k];
266 SequenceGeneratorBinom[n_, k_] := Binomial[n, k];
267 SequenceGeneratorBinomSquared[n_, k_] := Power[Binomial[n, k], 2];
268 SequenceGeneratorBinomSymmetric[n_, m_] := Binomial[n + m, m];
269 SequenceGeneratorDefaultFn[n_, k_] := 1;
270
271 SequenceGeneratorS2[n_, k_] := StirlingS2[n, k];
272
273 SequenceRowRangeDefaultTSeqV1[n_Integer] :=
274         Which[n < 0, {0, -1}, n == 0, {0, 0}, n >= 1, {1, n}]
275 SequenceRowRangeDefaultTSeqV2[n_Integer] :=
276         Which[n < 0, {0, -1}, n == 0, {0, 0}, n >= 1, {0, n - 1}]
277 SequenceRowRangeDefaultTSeqV3[n_Integer] :=
278         Which[n < 0, {0, -1}, n == 0, {0, 0}, n >= 1, {0, n}]
279 SequenceRowRangeDefaultFn[n_Integer] := If[n < 1, {0, -1}, {1, n}]
280
281 SequenceRowRangeS1[n_] := SequenceRowRangeDefaultTSeqV1[n]
282 SequenceRowRanges1[n_] := SequenceRowRangeDefaultTSeqV1[n]
283 SequenceRowRangeS2[n_] := SequenceRowRangeDefaultTSeqV1[n]
284 SequenceRowRangeE1[n_] := SequenceRowRangeDefaultTSeqV2[n]
285 SequenceRowRangeE2[n_] := SequenceRowRangeDefaultTSeqV2[n]
286 SequenceRowRangeBinom[n_] := SequenceRowRangeDefaultTSeqV3[n]
287 SequenceRowRangeBinomSquared[n_] := SequenceRowRangeDefaultTSeqV3[n]
288 SequenceRowRangeBinomSymmetric[n_] := SequenceRowRangeDefaultTSeqV3[n]
289
290 FullSequencesMetaData = {
291
292   {"SeqIDKey:S1:",
293     LongSeqID          -> "StirlingS1",
294     ShortSeqId         -> "S1",
295     Desc               -> "Stirling numbers of the first kind " <>
296                           "(unsigned triangle)",
297     NumInputs          -> 2,
298     StartRowIndex      -> 0,
299     AllowZeros         -> False,
```

```
300       GeneratorFunction -> SequenceGeneratorS1,
301       RowRangeFunction  -> SequenceRowRangeS1,
302       FormulaDisplay    -> "S1(``, ``)",
303       LatexDisplay      -> "\\genfrac{``}{``}",
304       Parameters        -> Null,
305       Comments          -> {"Sequence has OGF ... "},
306       OEISRefs          -> {},
307       PrintRefs         -> {"See \\S 6.1 of the Concrete Mathematics " <>
308                             "book and \\S 26.8 in the NIST Handbook."},
309       WebRefs           -> {"MathWorld: StirlingNumber.html"},
310       OtherRefs         -> {},
311       RevisionInfo      -> "",
312       Implementation    -> "",
313       ImplNotes         -> "",
314       OtherMetaData     -> Null
315     },
316
317     {"SeqIDKey:s1:",
318       LongSeqID         -> "Stirlings1",
319       ShortSeqId        -> "s1",
320       Desc              -> "Stirling numbers of the first kind " <>
321                            "(signed triangle)",
322       NumInputs         -> 2,
323       StartRowIndex     -> 0,
324       AllowZeros        -> False,
325       GeneratorFunction -> SequenceGenerators1,
326       RowRangeFunction  -> SequenceRowRanges1,
327       FormulaDisplay    -> "s(``, ``)",
328       LatexDisplay      -> "\\genfrac{``}{``}",
329       Parameters        -> Null,
330       Comments          -> {"Sequence has OGF ... "},
331       OEISRefs          -> {},
332       PrintRefs         -> {},
333       WebRefs           -> {},
334       OtherRefs         -> {},
335       RevisionInfo      -> "",
336       Implementation    -> "Standard Mathematica function (StrlingS1)",
337       ImplNotes         -> "",
338       OtherMetaData     -> Null
339     },
340
341     {"SeqIDKey:S2:",
342       LongSeqID         -> "StirlingS2",
343       ShortSeqId        -> "S2",
344       Desc              -> "Stirling numbers of the second kind",
345       NumInputs         -> 2,
346       StartRowIndex     -> 0,
347       AllowZeros        -> False,
348       GeneratorFunction -> SequenceGeneratorS2,
349       RowRangeFunction  -> SequenceRowRangeS2,
350       FormulaDisplay    -> "S(``, ``)",
351       LatexDisplay      -> "\\genfrac{``}{``}",
352       Parameters        -> Null,
353       Comments          -> {"Sequence has OGF ... "},
354       OEISRefs          -> {},
355       PrintRefs         -> {"See \\S 6.1 of the Concrete Mathematics " <>
356                             "book and \\S 26.8 in the NIST Handbook."},
357       WebRefs           -> {},
358       OtherRefs         -> {},
359       RevisionInfo      -> "",
360       Implementation    -> "Standard Mathematica function (StrlingS2)",
361       ImplNotes         -> "",
362       OtherMetaData     -> Null
363     },
364
365     {"SeqIDKey:E1:",
366       LongSeqID         -> "EulerianE1",
```

```
367       ShortSeqId          -> "E1",
368       Desc                -> "Triangle of the first-order Eulerian numbers",
369       NumInputs           -> 2,
370       StartRowIndex       -> 0,
371       AllowZeros          -> False,
372       GeneratorFunction   -> SequenceGeneratorE1,
373       RowRangeFunction    -> SequenceRowRangeE1,
374       FormulaDisplay      -> "",
375       LatexDisplay        -> "",
376       Parameters          -> Null,
377       Comments            -> {""},
378       OEISRefs            -> {},
379       PrintRefs           -> {"See \\S 6.2 of the Concrete Mathematics " <>
380                               "book and \\S 26.14 in the NIST Handbook."},
381       WebRefs             -> {},
382       OtherRefs           -> {},
383       RevisionInfo        -> "",
384       Implementation      -> "",
385       ImplNotes           -> "See also EulerianE1 in the " <>
386                              "RISC Stirling.m package",
387       OtherMetaData       -> Null
388     },
389
390     {"SeqIDKey:E2:",
391       LongSeqID           -> "EulerianE2",
392       ShortSeqId          -> "EulerianE2",
393       Desc                -> "Triangle of the second-order Eulerian numbers",
394       NumInputs           -> 2,
395       StartRowIndex       -> 0,
396       AllowZeros          -> False,
397       GeneratorFunction   -> SequenceGeneratorE2,
398       RowRangeFunction    -> SequenceRowRangeE2,
399       FormulaDisplay      -> "",
400       LatexDisplay        -> "",
401       Parameters          -> Null,
402       Comments            -> {""},
403       OEISRefs            -> {},
404       PrintRefs           -> {"See \\S 6.2 of the Concrete Mathematics book."},
405       WebRefs             -> {},
406       OtherRefs           -> {},
407       RevisionInfo        -> "",
408       Implementation      -> "",
409       ImplNotes           -> "See also EulerianE2 in the " <>
410                              "RISC Stirling.m package",
411       OtherMetaData       -> Null
412     },
413
414     {"SeqIDKey:Binom:",
415       LongSeqID           -> "Binomial",
416       ShortSeqId          -> "Binom",
417       Desc                -> "Binomial coefficients",
418       NumInputs           -> 2,
419       StartRowIndex       -> 0,
420       AllowZeros          -> False,
421       GeneratorFunction   -> SequenceGeneratorBinom,
422       RowRangeFunction    -> SequenceRowRangeBinom,
423       FormulaDisplay      -> "Binom(``, ``)",
424       LatexDisplay        -> "\binom{``}{``}",
425       Parameters          -> Null,
426       Comments            -> {},
427       OEISRefs            -> {},
428       PrintRefs           -> {},
429       WebRefs             -> {},
430       OtherRefs           -> {},
431       RevisionInfo        -> "",
432       Implementation      -> "Standard Mathematica function (Binomial)",
433       ImplNotes           -> "",
```

```
434       OtherMetaData      -> Null
435     },
436
437     {"SeqIDKey:BinomPow2:",
438       LongSeqID          -> "BinomialSquared",
439       ShortSeqId         -> "Binom2",
440       Desc               -> "Squares of the binomial coefficients",
441       NumInputs          -> 2,
442       StartRowIndex      -> 0,
443       AllowZeros         -> False,
444       GeneratorFunction  -> SequenceGeneratorBinomSquared,
445       RowRangeFunction   -> SequenceRowRangeBinomSquared,
446       FormulaDisplay     -> "",
447       LatexDisplay       -> "",
448       Parameters         -> Null,
449       Comments           -> {},
450       OEISRefs           -> {},
451       PrintRefs          -> {},
452       WebRefs            -> {},
453       OtherRefs          -> {},
454       RevisionInfo       -> "",
455       Implementation     -> "Standard Mathematica function (Binomial)",
456       ImplNotes          -> "",
457       OtherMetaData      -> Null
458     },
459
460     {"SeqIDKey:BinomSym:",
461       LongSeqID          -> "BinomialSymmetric",
462       ShortSeqId         -> "BinomSym",
463       Desc               -> "Binomial coefficients defined over " <>
464                             "symmetric index inputs. " <>
465                             "(see known generating functions)",
466       NumInputs          -> 2,
467       StartRowIndex      -> 0,
468       AllowZeros         -> False,
469       GeneratorFunction  -> SequenceGeneratorBinomSymmetric,
470       RowRangeFunction   -> SequenceRowRangeBinomSymmetric,
471       FormulaDisplay     -> "B(`1`+`2`, ``1)",
472       LatexDisplay       -> "\binom{`1`+`2`}{`1`}",
473       Parameters         -> Null,
474       Comments           -> {},
475       OEISRefs           -> {},
476       PrintRefs          -> {},
477       WebRefs            -> {},
478       OtherRefs          -> {},
479       RevisionInfo       -> "",
480       Implementation     -> "Standard Mathematica function (Binomial)",
481       ImplNotes          -> "",
482       OtherMetaData      -> Null
483     },
484
485     {"SeqIDKey:Example:",
486       LongSeqID          -> "Long identifier string for the sequence",
487       ShortSeqId         -> "Short ID string",
488       Desc               -> "Description of the sequence",
489       NumInputs          -> 2, (* number of input variables *)
490       StartRowIndex      -> 0, (* starting row for triangular sequences *)
491       AllowZeros         -> False,
492       GeneratorFunction  -> SequenceGeneratorDefaultFn,
493       RowRangeFunction   -> SequenceRowRangeDefaultFn,
494       FormulaDisplay     -> "Display string for printing " <>
495                             "local Mathematica notebook formulas",
496       LatexDisplay       -> "Display string for LaTeX outputs",
497       Parameters         -> Null, (* parameters for the sequence *)
498       Comments           -> {},
499       OEISRefs           -> {"List of related OEIS sequence numbers"},
500       PrintRefs          -> {"List of relevant print references"},
```

```
501       WebRefs             -> {"List of relevant website references"},
502       OtherRefs           -> {"Other relevant references for the sequence"},
503       RevisionInfo        -> "",
504       Implementation      -> "",
505       ImplNotes           -> "",
506       OtherMetaData       -> Null (* possibly for future use *)
507     }
508
509   };
510
511   (*************************************************************************)
512   (**** : Implementation of some non-standard triangular sequence     : ****)
513   (**** : functions defined by triangular recurrence relations:        ****)
514   (*************************************************************************)
515
516   Unprotect[SeqFnS1, SeqFnS2, SeqFnE1, SeqFnE2, GenTRec];
517
518   GenTRec[n_Integer, k_Integer,
519           alpha_, beta_, gamma_, alpha2_, beta2_, gamma2_] :=
520           GenTRecLocal[n, k, alpha, beta, gamma, alpha2, beta2, gamma2];
521
522   GenTRecLocal[n_Integer, k_Integer, a_, b_, g_, a2_, b2_, g2_] :=
523   Which[
524       n < 0 || k < 0, 0,
525       n == 0, KroneckerDelta[n, k],
526       n >= 0, GenTRecLocal[n, k, a, b, g, a2, b2, g2] =
527                 (a n + b k + g) *
528                 GenTRecLocal[n - 1, k, a, b, g, a2, b2, g2] +
529                 (a2 n + b2 k + g2) *
530                 GenTRecLocal[n - 1, k - 1, a, b, g, a2, b2, g2]
531   ];
532
533   SeqFnS1[n_Integer, k_Integer] := GenTRec[n, k, 1, 0, -1, 0, 0, 1];
534   Format[SeqFnS1[n_, k_], TraditionalForm] := Subscript["S", 1][n, k];
535
536   SeqFnS2[n_Integer, k_Integer] := GenTRec[n, k, 0, 1, 0, 0, 0, 1];
537   Format[SeqFnS2[n_, k_], TraditionalForm] := Subscript["S", 2][n, k];
538
539   SeqFnE1[n_Integer, k_Integer] := GenTRec[n, k, 0, 1, 1, 1, -1, 0];
540   Format[SeqFnE1[n_, k_], TraditionalForm] := Subscript["E", 1][n, k];
541
542   SeqFnE2[n_Integer, k_Integer] := GenTRec[n, k, 0, 1, 1, 2, -1, -1];
543   Format[SeqFnE2[n_, k_], TraditionalForm] := Subscript["E", 2][n, k];
544
545   Protect[SeqFnS1, SeqFnS2, SeqFnE1, SeqFnE2, GenTRec];
546
547   End[]  (* Private *)
548
549
550   GenTRec::usage =
551       "Paremetrized triangular sequence definition used " <>
552       "to define several special case triangles, including the " <>
553       "Stirling numbers of the first and second kinds, and the " <>
554       "Eulerian numbers of both orders.";
555
556   SeqFnS1::usage =
557       "Computes the unsigned Stirling numbers of the first kind " <>
558       "for n,k \[GreaterEqual] 0.\n" <>
559        "See the key \"S1\" from the GuessSequenceData package.";
560   SeqFnS2::usage =
561       "Computes the Stirling numbers of the second kind " <>
562       "for n,k \[GreaterEqual] 0.\n" <>
563        "See the key \"S2\" from the GuessSequenceData package.";
564   SeqFnE1::usage =
565       "Computes the first-order Eulerian numbers " <>
566       "for n,k \[GreaterEqual] 0.\n" <>
567       "See the key \"E1\" from the GuessSequenceData package.";
```

```
568 SeqFnE2::usage =
569       "Computes the second-order Eulerian numbers " <>
570       "for n,k \[GreaterEqual] 0.\n" <>
571       "See the key \"E2\" from the GuessSequenceData package.";
572
573 EndPackage[]
574
575 (*******************************************************************)
576 (*******************************************************************)
577 (*******************************************************************)
578 (*******************************************************************)
```